INSTITUTO TECNOLOGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY

CAMPUS MORELOS

MAESTRIA EN CIENCIAS COMPUTACIONALES

T E S I S

PLANIFICACION DE TAREAS FLEXIBLES PARA
SISTEMAS DE TIEMPO REAL

P R E S E N T A

JORGE EDUARDO APARICIO MAYORGA

CUERNAVACA, MOR. AGOSTO 1998

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Morelos

MAESTRÍA EN CIENCIAS COMPUTACIONALES

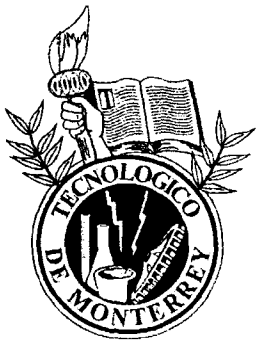
T E S I S

PLANIFICACIÓN DE TAREAS FLEXIBLES PARA SISTEMAS DE TIEMPO REAL

PRESENTA:

JORGE EDUARDO APARICIO MAYORGA

CUERNAVACA, MOR. AGOSTO 1998



Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Morelos

Maestría en Ciencias Computacionales

Planificación de Tareas Flexibles para Sistemas de Tiempo Real

Aprobó:

Comité Revisor

Asesor: Dr. Roberto Valdivia Beutelspacher
Director de Ingeniería Computacional
ITESM, Campus Morelos.

Asesor: Dr. Luis Enrique Sucar Succar
Director Académico del Departamento de Computación
ITESM, Campus Morelos.

Revisor: Dr. Víctor Hugo Zárate Silva
Director Académico del Departamento de Electrónica
ITESM, Campus Morelos.

Revisor: Dr. José Torres Jiménez
Profesor-Investigador
ITESM, Campus Morelos.

Dedicatorias

A mis abuelitos Min(✞), Trini(✞), Jesús(✞) y Luz María, semillas de una familia extraordinaria.

A mis padres Jesús y Ma. Elena, por modelar mi vida con su ejemplo de responsabilidad, amor y éxito.

A mis hermanos Jesús Alberto y Erick Benjamín, para que nunca perdamos el cariño que nos une.

A Rocío, por su amor incondicional a pesar del tiempo y la distancia.

A todos los amo.

Agradecimientos

A mi asesor y amigo Dr. Roberto Valdivia, por ayudarme cuando estaba confundido y por su permanente disposición para dirigir este proyecto.

A mi asesor Dr. Luis Enrique Sucar, por su tiempo ilimitado y los comentarios que enriquecieron esta tesis.

A los Doctores José Torres y Víctor Hugo Zárate por sus observaciones en la revisión de esta tesis.

A todos mis tíos y primos, por hacerme feliz al tener una familia como la nuestra.

A mis compañeros de casa, Carlos Mariño, Chube, Martín Molina y Carlos Mena, por hacer de mi estancia en Cuernavaca una época inolvidable.

A todos mis amigos, por los momentos compartidos. Siempre los tendré presentes en mis pensamientos.

Al Escultismo, por hacerme un hombre responsable y enseñarme a amar a Dios, a mi México y a mi familia.

Al pueblo y gobierno de México, así como a la Fundación TELMEX, por su apoyo económico durante mis estudios de maestría.

RESUMEN

En la práctica, los sistemas de tiempo real están compuestos por una mezcla de tareas periódicas, aperiódicas y esporádicas cuyos plazos de respuesta son muy importantes, ya que no cumplirlos podría ser causa de resultados catastróficos, tales como grandes perjuicios económicos o la pérdida de vidas humanas. En una sobrecarga, es decir, cuando el tiempo disponible para la ejecución de las tareas del sistema no es suficiente para cumplir los plazos de respuesta de todas las tareas, algunas de ellas invariablemente perderán estos plazos.

La computación flexible es una manera de mantener la estabilidad de un sistema durante una sobrecarga. En general, la computación flexible se refiere a aquellos procedimientos de solución de problemas que permiten un compromiso paulatino entre la calidad de los resultados y la asignación de recursos, tales como tiempo, memoria e información. Para aquellas aplicaciones que toleran el uso de resultados aproximados, utilizando este enfoque se puede construir un sistema de tiempo real como un conjunto de tareas flexibles. Éstas se definen como aquellas tareas que pueden ser interrumpidas antes de ser terminadas totalmente produciendo resultados aproximados, pero útiles, cuya calidad (medida a través de funciones de error) depende del tiempo que hayan permanecido en ejecución.

En esta tesis se desarrolla un método de Planificación de Tareas periódicas, aperiódicas y esporádicas Flexibles para un sistema de un solo procesador que permite mantener la estabilidad del sistema en situaciones de sobrecarga, el cual se denominó método PTF. Este método es producto de la integración y adaptación de las ideas de dos algoritmos de planificación de tareas flexibles existentes: LAT y NORA. El método PTF fue evaluado de manera empírica, mediante un ambiente de simulación construido exprofeso. La simulación permitió observar el comportamiento del método PTF en diferentes circunstancias y comparar los planes producidos por él contra otros cuatro algoritmos de planificación, para los cuales también se construyeron ambientes de simulación. Los resultados de la simulación indican que el método PTF es un algoritmo de planificación de tareas flexibles que, en presencia de sobrecargas, produce niveles de error menores en la ejecución de las tareas flexibles que los algoritmos contra los que fue comparado.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE TABLAS

Capítulo 1

INTRODUCCIÓN

1.1 Motivación

Cada día, más y más funciones que antes eran realizadas por operadores humanos son llevadas a cabo por computadoras digitales. Muchas de estas funciones se refieren a procesos físicos de importancia vital para la sociedad que deben ser controlados permanentemente y que involucran importantes restricciones de confiabilidad y de tiempo (temporales). Los sistemas computacionales que implantan estas funciones se conocen como sistemas de tiempo real. Algunos ejemplos de sistemas de tiempo real son: sistemas de control de procesos de manufactura, sistemas de control de vuelos, sistemas de control de automóviles, sistemas militares y sistemas de control de plantas nucleares, entre otros [Burns y Wellings, 90] [Ramamrithan y Stankovic, 94] [Shin y Ramanathan, 94].

Como se mencionó previamente los sistemas de tiempo real tienen restricciones temporales, a diferencia de los sistemas computacionales tradicionales. Sus resultados deben satisfacer dichas restricciones para poder ser considerados como soluciones. En otras palabras, la correctez de un sistema de tiempo real depende no sólo de los cálculos, sino del tiempo en que se producen los resultados. Dependiendo de la aplicación de que se trate, si un sistema de tiempo real no cumple con

sus restricciones temporales podrían producirse consecuencias catastróficas, como son la pérdida de vidas humanas o grandes perjuicios económicos. Por ejemplo, si el sistema de control de una planta nuclear no puede cumplir sus restricciones temporales podría producirse un accidente nuclear similar al de Chernobyl, Ucrania (abril de 1986). O bien, pensando en un sistema de control de un proceso de manufactura, el hecho de no cumplir sus restricciones temporales podría traducirse en productos incompletos, inservibles o de mala calidad.

Las restricciones temporales de los sistemas de tiempo real usualmente se traducen en plazos de respuesta *(deadlines)* impuestos sobre las tareas que lo componen. Para determinar adecuadamente los momentos en que deben ejecutarse las tareas de un sistema de tiempo real se recurre a un planificador de tareas *(scheduler)*. Un planificador de tareas es un algoritmo implantado en el *kernel* de un sistema operativo de tiempo real, cuya labor es hacer una programación de tareas en el procesador, con respecto al tiempo, que cumpla los plazos de respuesta de las tareas individuales. Se han desarrollado varios algoritmos de planificación de tareas en tiempo real (por ejemplo, el algoritmo de "razón monotónica" [Liu y Layland, 73] y el algoritmo de "primero el plazo más próximo"[Klein et al., 93]), los cuales son útiles para diferentes aplicaciones dependiendo de las características de éstas.

Un sistema de tiempo real puede trabajar erróneamente cuando ocurren errores de *software*, *hardware*, o cuando no se cumplen todos los plazos de respuesta de las tareas. La pérdida de plazos de respuesta ocurre invariablemente cuando el sistema está sobrecargado, es decir, cuando el tiempo disponible no es suficiente para completar la ejecución de todas las tareas que requieren atención [Liu et al., 94].

El comportamiento deseable en un sistema de tiempo real en presencia de una sobrecarga es el de mantener la estabilidad del sistema. Esto quiere decir, que a pesar de que sea imposible cumplir los plazos de respuesta de todas las tareas, debe garantizarse que se cumplirán los plazos de aquellas tareas que sean más críticas [Sha et al., 94]. El enfoque de tolerancia a fallas[1] se ha encargado de buscar soluciones que permitan mantener la estabilidad del sistema no sólo en casos de sobrecarga, sino ante la presencia de errores de *software* y *hardware*. Desafortunadamente, la importancia de

---

[1] Tolerancia a fallas se define informalmente como la habilidad de un sistema de proporcionar el servicio esperado, aún en presencia de fallas en el sistema o en el ambiente.

cumplir las restricciones temporales y la tolerancia a fallas muchas veces entran en conflicto. Por ejemplo, una verificación de errores frecuente y la ejecución de rutinas de recuperación de errores complejas incrementa la tolerancia a fallas, pero también aumenta la posibilidad de que las tareas de las aplicaciones pierdan sus plazos de respuesta [Shin y Ramanathan, 94].

Un enfoque alternativo para lograr mantener la estabilidad del sistema cuando el tiempo es limitado es la *computación flexible*. La computación flexible es una nueva manera de ver la computación que surgió a fines de la década de los ochentas. Dicho concepto se refiere a los procedimientos de solución de problemas que permiten establecer un compromiso paulatino entre la calidad de los resultados y la asignación de recursos, tales como tiempo, memoria o información [Horvitz, 87].

Siendo el tiempo el recurso principal de un sistema de tiempo real [Shin y Ramanathan, 94], resulta interesante considerar la aplicación de algoritmos de computación flexible a los sistemas de tiempo real. La idea es evitar la pérdida de plazos de respuesta, de modo que cuando el sistema no disponga de tiempo suficiente para satisfacer los plazos de todas las tareas presentes, éstas sacrifiquen la calidad de sus resultados, reduciendo sus tiempos de ejecución. El objetivo es que aunque los resultados producidos por las tareas no sean precisos, sí tengan la calidad suficiente para mantener al sistema estable, es decir, cumpliendo con sus plazos de respuesta.

Las tareas a que hace referencia la computación flexible (cuando el recurso limitado es el tiempo) no son tareas tradicionales. Éstas son tareas flexibles (*anytime*), en el sentido de que pueden ser interrumpidas prematuramente y producir un resultado cuya calidad depende del tiempo que se hayan ejecutado [Liu et al., 94] [Grass, 96].

También se han desarrollado algoritmos de planificación para aplicaciones desarrolladas como tareas flexibles. A excepción del trabajo presentado en [Davis et al., 95], los trabajos sobre planificación de tareas flexibles existentes abarcan, de manera independiente, la planificación de actividad periódica y aperiódica [Chung et al., 90] [Shih y Liu, 92] [Liu et al., 94]. Sin embargo, en la práctica un sistema de tiempo real se compone de una mezcla de tareas periódicas, aperiódicas y esporádicas [Strosnider et al., 95].

1.2 Objetivo de la tesis

Por la importancia de mantener la estabilidad de los sistemas de tiempo real en la presencia de sobrecargas y la carencia de trabajos sobre planificación de conjuntos de tareas periódicas, aperiódicas y esporádicas flexibles, el objetivo principal de esta tesis es el desarrollo de un método de planificación de tareas periódicas, aperiódicas y esporádicas flexibles para un sistema uniprocesador que mantenga la estabilidad del sistema aún en presencia de sobrecargas.

Otros objetivos que se busca lograr en el desarrollo de esta tesis son:

- Estudiar diferentes enfoques de implantación de computación flexible cuando el recurso restringido es el tiempo.
- La implantación, a nivel de simulación, de un prototipo que permita evaluar el comportamiento del método desarrollado y compararlo contra otros algoritmos de planificación tradicionales.

1.3 Organización de la tesis

En el **Capítulo 2, Sistemas de tiempo real**, se describen aspectos fundamentales sobre sistemas de tiempo real, los cuales son utilizados en los capítulos posteriores, principalmente en el capítulo 4. Entre ellos: las nociones de tareas periódicas, aperiódicas y esporádicas; la estructura de los sistemas operativos de tiempo real; los mecanismos de interacción entre tareas; y algunas nociones relevantes sobre planificación de tareas en sistemas de tiempo real.

En el **Capítulo 3, Computación flexible**, se presenta el concepto de computación flexible y se explican los enfoques principales que se han desarrollado al respecto.

El **Capítulo 4, Planificación de tareas flexibles**, detalla el método de planificación desarrollado, al cual se denomina método PTF (método de Planificación de Tareas periódicas, aperiódicas y esporádicas Flexibles). Inicia describiendo los algoritmos en que se basa el método PTF, hasta llegar a su definición. Incluye secciones en las que se evalúa la complejidad temporal del método y los requerimientos básicos de soporte de sistema operativo.

El **Capítulo 5**, **Resultados**, presenta resultados obtenidos mediante una simulación, sobre el comportamiento del método PTF. También se incluye una comparación contra otros cuatro algoritmos de planificación no flexibles.

Finalmente, el **Capítulo 6**, **Conclusiones y trabajos futuros**, presenta las conclusiones generadas a partir de este trabajo de tesis y sus posibles extensiones.

Capítulo 2

SISTEMAS DE TIEMPO REAL

Al conjunto de un sistema computacional que controla un proceso físico, el cual produce estímulos que deben ser atendidos en tiempos críticos, y al propio proceso físico, se le conoce informalmente como sistema de tiempo real. La conveniencia de tener sistemas automatizados que permitan desarrollar eficientemente las operaciones que antes realizaban los humanos, y el rápido avance en el desarrollo del *hardware,* han sido factores que han motivado un acelerado desarrollo de los sistemas de tiempo real.

Este capítulo introduce nociones y conceptos básicos sobre sistemas de tiempo real a los cuales se hará referencia a lo largo de esta tesis. En la sección 2.1 se explican las características de los sistemas de tiempo real. La sección 2.2 explica la manera en que interactúan las tareas que componen un sistema de tiempo real y los problemas que pueden presentarse debido a estas interacciones. La sección 2.3 presenta los elementos y características de un sistema operativo de tiempo real. La sección 2.4 explica las diferentes técnicas que existen para que uno de los elemento más importantes de un sistema operativo de tiempo real, el planificador, realice su trabajo.

2.1 Conceptos fundamentales

2.1.1 Definiciones de sistema de tiempo real

Existen varias interpretaciones sobre el significado del término "sistemas de tiempo real", sin embargo, todas tienen en común la noción de tiempo de respuesta. Por ejemplo, [Ramamrithan y Stankovic, 94] definen los sistemas de tiempo real como:

"...aquellos sistemas en los que la correctez del sistema depende no sólo del resultado lógico de los cómputos, sino del tiempo en que se producen estos resultados."

En [Burns y Wellings, 90] se dice que un sistema de tiempo real es:

"cualquier actividad de procesamiento de información o sistema que tiene que responder a estímulos generados externamente dentro de un período de tiempo finito especificado."

Las aplicaciones que han encontrado los sistemas de tiempo real son variadas e incluyen: el control de plantas nucleares y de generación de energía eléctrica, control de procesos industriales, sistemas de control de vuelos, sistemas de defensa, control de procesos de manufactura, etc. [Burns y Wellings, 90] [Ramamrithan y Stankovic, 94] [Shin y Ramanathan, 94].

2.1.2 Características de los sistemas de tiempo real

Usualmente un sistema de tiempo real se forma por un proceso físico y un sistema de cómputo que lo controla. El sistema de cómputo se compone por un conjunto de tareas concurrentes, cooperantes y competidoras entre sí, las cuales deben cumplir restricciones temporales, de predecibilidad, de recursos, de precedencia y de confiabilidad. Normalmente, las tareas se activan en períodos regulares de tiempo y tienen tiempos límite de respuesta (plazos de respuesta), aunque la activación también puede darse en intervalos irregulares. En cada activación una tarea monitorea el estado del proceso físico, realiza algún cálculo y, si es necesario, envía comandos para cambiar o desplegar dicho estado [Shin y Ramanathan, 94].

2.1.2.1 Restricciones Temporales

Es común que en el desarrollo de programas de aplicación de sistemas de tiempo real los requerimientos temporales se reflejen como plazos en el inicio o término de las tareas. De acuerdo

con los efectos que se producen al no cumplirse los plazos impuestos sobre las tareas, los sistemas de tiempo real pueden dividirse en estrictos y flexibles.

Sistemas de tiempo real estrictos

Los sistemas de tiempo real estrictos, son aquellos sistemas en que es muy importante que las respuestas a los eventos, usualmente codificadas como tareas, ocurran dentro de los plazos especificados, pues podría ocurrir un daño catastrófico si se perdiese alguno de ellos [Burns y Wellings, 90], [Burns, 91]. La Figura 2.1 muestra la función de utilidad respecto al tiempo de una tarea con plazo de respuesta estricto, en la cual, al no cumplirse el plazo la utilidad es cero (aunque no ocurre daño).

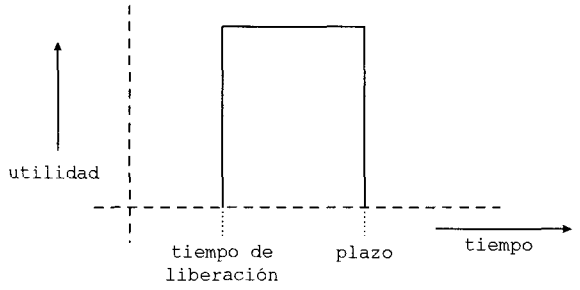


Figura 2.1 Utilidad de una tarea con plazo de respuesta estricto [Burns, 91].

La utilidad se refiere a la contribución que tiene el resultado de la tarea a los objetivos del sistema de tiempo real. El tiempo de liberación corresponde al momento en que la tarea está lista para ser ejecutada.

Sin embargo, la situación en un sistema de tiempo real estricto puede ser aún peor, produciéndose daño real, representado por una utilidad negativa, como resultado de iniciar una tarea antes de su tiempo de liberación o de no cumplir su plazo de respuesta, como se muestra en la Figura 2.2.

Sistemas de tiempo real flexibles

Los sistemas de tiempo real flexibles son aquellos en que los tiempos de respuesta son importantes, pero en los que el sistema no dejará de funcionar correctamente si ocasionalmente no se cumplen

algunos plazos de respuesta [Burns y Wellings, 90] [Burns, 91]. La Figura 2.3 presenta la función de utilidad típica de una tarea con plazo de respuesta flexible.

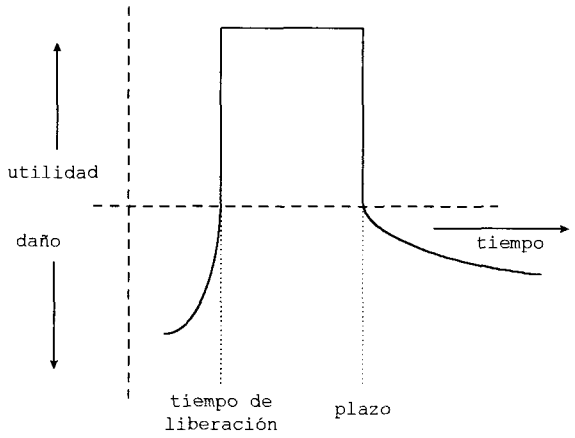


Figura 2.2 Utilidad de una tarea con plazo de respuesta estricto en la que ocurre daño [Burns, 91].

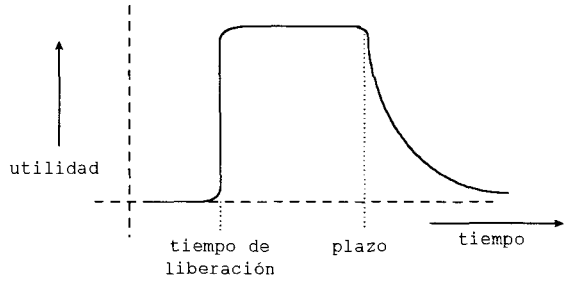


Figura 2.3 Utilidad de una tarea con plazo de respuesta flexible [Burns, 91].

En las aplicaciones reales, las tareas que componen los sistemas de tiempo real muestran una combinación de tareas con plazos de respuesta estrictos y flexibles.

2.1.2.1.1 Clasificación de las tareas de acuerdo a su intervalo de activación

El objetivo de cumplir con las restricciones temporales de un sistema de tiempo real, se transforma en el de planificar sus tareas, las cuales pueden ser de tres tipos de acuerdo a su intervalo de activación: periódicas, aperiódicas y esporádicas [Burns, 91].

Tareas periódicas

Son tareas invocadas en intervalos regulares de tiempo. Por ejemplo, la actividad de monitorear reiteradamente el estado de un sensor en un sistema de vigilancia, puede implantarse como una tarea periódica. Las tareas periódicas se caracterizan por tener:

- *Período.* La longitud del intervalo de tiempo entre cada invocación de la misma tarea.
- *Plazo de respuesta.* Instante de tiempo en que la tarea ya debe haber terminado. Usualmente el plazo de respuesta es igual al período de la tarea.
- *Tiempo de ejecución por período.* Se refiere al tiempo de CPU necesario para ejecutar la tarea. Normalmente el tiempo de ejecución se da en términos de una medida promedio y/o de un tiempo de ejecución en el peor de los casos [Burns, 91].

Tareas aperiódicas

Son tareas recurrentes que se activan sólo cuando ocurren ciertos eventos que usualmente corresponden a eventos aleatorios emitidos por acciones externas al sistema y que por tanto no arriban en períodos regulares. Por ejemplo, la activación manual de un interruptor. Las tareas aperiódicas pueden llegar con intervalos entre invocaciones de tamaño arbitrariamente pequeño e irregular. Es común que la activación de tareas aperiódicas esté descrita por una función de densidad probabilística (generalmente una distribución Poisson). Esta distribución permite generar llegadas de eventos externos en ráfaga, sin excluir cualquier concentración posible de actividad aperiódica. Por tanto, no es posible hacer un análisis del peor de los casos (existe una posibilidad finita, es decir, distinta a cero, de que ocurra cualquier número de eventos aperiódicos en un instante o período). Por consiguiente, las tareas aperiódicas no pueden tener plazos de respuesta estrictos, sólo pueden tener plazos de respuesta flexibles [Burns, 91].

Tareas esporádicas

Las tareas esporádicas son un caso especial de las tareas aperiódicas. Cuando existe un tiempo mínimo entre la invocación de dos ocurrencias de una misma tarea aperiódica, se dice que la tarea es esporádica. En el caso de las tareas esporádicas, sí es posible hacer un análisis del peor de los casos, pues se conoce su tiempo mínimo de activación entre dos ocurrencias. Las tareas esporádicas pueden tener o no plazos de respuesta estrictos y son usadas en el manejo de eventos tales como la atención de interrupciones de los dispositivos y la respuesta a las entradas del usuario [Burns, 91].

2.1.2.2 Restricciones de Predecibilidad

Puesto que muchos de los sistemas en tiempo real soportan actividades críticas, su comportamiento debe ser predecible, es decir, debe ser posible demostrar en la etapa de diseño que todas las restricciones temporales de las tareas de la aplicación serán cumplidas mientras se satisfagan ciertas suposiciones sobre el sistema.

Es importante hacer notar que sólo es posible garantizar el cumplimiento de los plazos de respuesta cuando las características de las tareas, tales como tiempo de ejecución y tiempo de llegada, son conocidos *a priori*. En la práctica es difícil obtener información exacta sobre las características de las tareas, por lo que los valores del peor de los casos se asumen o se derivan a partir de simulaciones exhaustivas, lo cual no excluye la posibilidad de que sean incorrectos.

2.1.2.3 Restricciones de Precedencia

Es común que una tarea requiera resultados de otra u otras tareas antes de poder iniciar su ejecución [Shin y Ramanathan, 94]. Si una tarea *Ti* recibe como datos de entrada los resultados producidos por una tarea *Tj*, entonces se dice que *Tj* precede a *Ti*. Las restricciones de precedencia pueden ser modeladas con un grafo dirigido en el que los vértices representan las tareas y los arcos representan la relación de precedencia, como puede apreciarse en la Figura 2.4. Por ejemplo, según la figura, las tareas *b* y *c* deben terminar antes de que la tarea *f* pueda comenzar su ejecución.

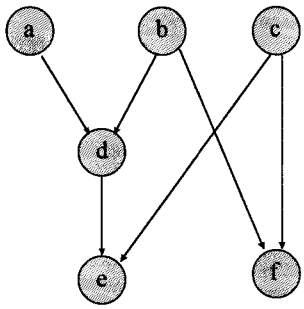


Figura 2.4 Restricciones de precedencia de una aplicación de tiempo real.

2.1.2.4 Restricciones de Recursos

Las tareas de un sistema de tiempo real usualmente se ejecutan concurrentemente, es decir, se ejecutan en un intervalo de tiempo común en un mismo procesador. La concurrencia comúnmente se

debe a que el número de tareas por ejecutar es mayor al número de procesadores disponibles, por lo que varias tareas deben compartir un mismo procesador. Con ello, dichas tareas podrían requerir compartir alternadamente algunos recursos comunes, por ejemplo el procesador [Stallings, 97]. Además del procesador, una tarea podría requerir acceder a otros recursos del sistema. Ejemplos de recursos son: dispositivos de E/S, redes de comunicación, estructuras de datos, archivos y bases de datos. Cuando varias tareas se ejecutan concurrentemente, en muchos casos es necesario que exista exclusión mutua entre ellos, es decir, la capacidad para impedir a todas las otras tareas, el acceso a un recurso mientras a alguna le ha sido asignado dicho recurso [Shin y Ramanathan, 94].

En un ambiente de procesamiento concurrente con restricciones de recursos no puede predecirse el tiempo que las tareas tardarán en ejecutarse totalmente; ello depende de las actividades de otras tareas, de la manera en que el sistema operativo maneja las interrupciones y de las políticas de planificación [Stallings, 97].

2.1.2.5 Restricciones de Confiabilidad

En los sistemas de tiempo real estrictos, los requerimientos temporales deben cumplirse aún bajo condiciones hostiles en el ambiente, como pueden ser sobrecargas, fallas o errores. No cumplir con un plazo de respuesta estricto puede ocasionar consecuencias desastrosas.

Una sobrecarga se produce cuando el sistema de tiempo real no cuenta con el tiempo suficiente para cumplir con todas las restricciones temporales de las tareas que deben ejecutarse [Baruah y Haritsa, 97]. Las causas que pueden dar origen a una sobrecarga son: modificaciones al proceso físico o avalanchas de fallas en el sistema de tiempo real. Sus consecuencias van desde la generación de productos incompletos en sistemas de manufactura, hasta la ocurrencia de desastres con grandes pérdidas económicas y humanas en sistemas que desempeñan funciones vitales para la sociedad.

Por la naturaleza de las aplicaciones, los sistemas de tiempo real normalmente deben ser tolerantes a fallas. La tolerancia a fallas hace referencia a la capacidad de un sistema de continuar en funcionamiento a pesar de la ocurrencia de una falla. Un aspecto importante de la tolerancia a fallas es la estabilidad. Un sistema de tiempo real es estable si, en los casos que es imposible cumplir todos los plazos de respuesta de las tareas, se cumplen los plazos de las tareas más críticas y de mayor

prioridad, incluso si no se cumplen los de alguna tarea menos crítica [Stallings, 97][Burns y Wellings, 90].

Un sistema puede presentar tolerancia a fallas a varios niveles. El nivel de tolerancia requerido depende de cada aplicación particular. Los niveles de tolerancia a fallas posibles son [Burns y Wellings, 90]:

- *Tolerancia a fallas completa.* Al presentarse una falla, el sistema de tiempo real sigue operando sin pérdida de funcionalidad o desempeño por un período limitado de tiempo.
- *Degradación paulatina.* El sistema sigue operando, aunque con una degradación paulatina de su funcionalidad o desempeño mientras se recupera o repara la falla.
- *Estado seguro.* Al ocurrir una falla, el sistema se detiene temporalmente en un estado que mantiene su integridad y/o la seguridad de los usuarios del sistema.

Mientras el sistema de tiempo real se comporta de acuerdo con su especificación se dice que proporciona un servicio correcto. La presencia de una falla lleva al sistema a un estado de servicio incorrecto, el cual deberá ser detectado, regresando al estado de servicio correcto a través de un proceso de recuperación o reparación (Figura 2.5).

Figura 2.5 Estados de servicio de un sistema de tiempo real.

Algunas medidas que ayudan a determinar en qué grado un sistema de tiempo real provee un servicio correcto son:

- *Confiabilidad.* La probabilidad de que un sistema funcione sin fallas durante un período de tiempo particular dado que funcionaba correctamente en un tiempo inicial [Myers, 76]. En otras palabras, la capacidad de brindar continuidad de servicio correcto.
- *Disponibilidad.* Se refiere a la probabilidad de que un sistema esté operando de acuerdo a su especificación en un instante de tiempo específico [Tamer y Valduriez, 91].
- *Seguridad (safety).* Inmunidad del sistema a las condiciones que podrían causar muertes, daños, pérdida de equipo o daño ambiental [Burns y Wellings, 90].

- *Dependencia (dependability).* La propiedad de un sistema de ser confiable y seguro. Un sistema con un alto nivel de dependencia es aquel que está protegido contra fallas intencionales (sabotaje) y brinda sus servicios de manera continua sin que se presenten consecuencias catastróficas [Burns y Wellings, 90]. Dependencia involucra confiabilidad, disponibilidad y seguridad.
- *Estabilidad.* Como se mencionó anteriormente, un sistema de tiempo real es estable si, en los casos que es imposible cumplir todos los plazos de respuesta de las tareas, el sistema cumple los plazos de las tareas más críticas y de mayor prioridad, incluso si no se cumplen los de alguna tarea menos crítica [Stallings, 97].

2.2 Interacción entre tareas

En general, las tareas presentes un sistema de cómputo pueden cooperar o no en la consecución de un objetivo común y/o competir por el uso de los recursos del sistema [Stallings, 97].

2.2.1 Competición por recursos

La competición por recursos es uno de los problemas que provoca la ejecución concurrente de varias tareas. Este problema se presenta cuando dos o más tareas que no tienen comunicación ni intercambian información pretenden acceder al mismo recurso. La ejecución de una tarea puede afectar el comportamiento de sus competidores, haciendo su ejecución más lenta e inclusive provocándoles muerte por inanición. Por otro lado, puesto que las tareas no hacen uso exclusivo de los recursos, al liberarlos deben dejarlos como si no lo hubiesen usado. Las situaciones que deben resolverse por la competición de recursos son [Stallings, 97]:

a) *Exclusión mutua.* Una sección crítica es un grupo de instrucciones de una tarea en que se hace uso de un recurso en particular. El requerimiento de exclusión mutua es el de asegurar que sólo una de las tareas que compiten por un mismo recurso, esté en su sección crítica a la vez. Si no hay tareas que estén ejecutando su sección crítica, cualquier tarea que necesite usar el recurso compartido puede hacerlo sin tener que esperar.

b) *Interbloqueos*. Deben evitarse situaciones en que un grupo de tareas, digamos T_1, T_2, T_3, ..., T_n, que tienen asignados recursos, digamos R_1, R_2, R_3, ..., R_n, respectivamente, hagan solicitudes que nunca podrán ser satisfechas. Por ejemplo, T_1 que posee el recurso R_1, solicita el recurso R_2 que posee T_2. T_1 tiene que esperar. Por su parte T_2 solicita el recurso R_3 que posee T_3, por lo que T_2 debe esperar. Siguiendo este patrón sucesivamente, llegamos al punto en que T_{n-1} solicita el recurso R_n que posee T_n . Finalmente, si T_n solicita el recurso R_1, que posee T_1, se obtiene una situación en la que varias tareas están bloqueados esperando por recursos que tienen mutuamente asignados y que no serán liberados. La Figura 2.6 presenta un grafo de espera *G(V,E)* mostrando una situación de interbloqueo entre cuatro tareas. El conjunto V de nodos del grafo corresponde a las tareas del sistema de tiempo real, y existe una arista $T_i \rightarrow T_j$ en el grafo de espera si la tarea T_i está esperando la liberación de un recurso que posee T_j.

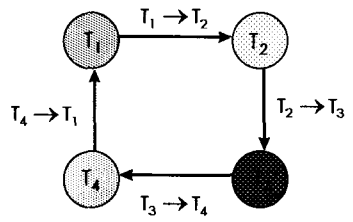


Figura 2.6 Grafo de espera *G(V,E)* mostrando un interbloqueo.

Los interbloqueos pueden clasificarse dependiendo del tipo de solicitud de recursos que pueden hacer las tareas. El ejemplo mostrado representa el caso más simple de interbloqueo conocido como *modelo de solicitud de una sola unidad (single-unit request model)*, el cual se caracteriza porque las tareas sólo pueden solicitar una unidad de un recurso a la vez. En general, existen cuatro modelos de interbloqueo [Singhal y Shivaratri, 94]:

Modelo de solicitud *AND (AND request model)*. En este modelo, una tarea puede solicitar simultáneamente múltiples unidades de un mismo recurso, y permanece bloqueada hasta que le son garantizadas todas las unidades solicitadas. El modelo de solicitud de una sola unidad es un caso particular de este modelo.

Modelo de solicitud *OR*. En este modelo una tarea puede solicitar simultáneamente múltiples unidades de un mismo recurso, y permanece bloqueada hasta que le es garantizado al menos uno de ellos. El modelo de solicitud de una sola unidad también es un caso particular de este modelo.

Modelo de solicitud *AND-OR*. Este modelo es una generalización de los dos modelos anteriores en el que las solicitudes de recursos se especifican utilizando predicados cuyas variables son los recursos. Por ejemplo, la solicitud R_1 AND (R_2 OR R_3) puede ser satisfecha con R_1 AND R_2 o con R_1 y R_3.

Modelo de solicitud P de Q. En este modelo, una tarea puede solicitar simultáneamente Q recursos, y permanece bloqueada hasta que le son garantizados cualesquiera P de esos recursos.

c) *Inanición*. Una situación de inanición se presenta cuando un grupo de tareas que compiten por un recurso R_1, digamos P_1 y P_2, intercambian alternadamente el control sobre él, provocando que una tarea P_3 que también compite por R_1 quede esperando indefinidamente aunque no se presente una situación de interbloqueo.

d) *Inversión de prioridades*. La inversión de prioridades es un problema muy serio en los sistemas de tiempo real, pues afecta la predecibilidad de los sistemas [Sha et al., 90]. Se trata de un fenómeno en el que una tarea de alta prioridad es bloqueada durante un tiempo impredecible por tareas de prioridad menor. La Figura 2.7 muestra un ejemplo de inversión de prioridades. Sean H, M y L tres tareas de alta, media y baja prioridad respectivamente. El diagrama representa la ejecución de las tareas H, M y L, las cuales usan las operaciones P y V para control de concurrencia a través de semáforos[1]. Cuando la línea que representa cada tarea está en el nivel superior indica que la tarea está en ejecución. El área sombreada representa una sección crítica. En t_2, H necesita un recurso que L tiene asignado, por lo tanto H es bloqueada por L. En t_3, M solicita ser ejecutada, y como su prioridad es mayor que la de L, L es removida del procesador para permitir la ejecución de M, lo cual retrasa aún más la espera de H. La duración del bloqueo es impredecible, pues H tendrá que esperar a que se atiendan todas las tareas de prioridad media

[1] Un semáforo es un mecanismo de control de concurrencia compuesto por una variable entera, usada para realizar señalización entre procesos, y tres operaciones atómicas sobre dicha variable: inicializar, *wait* (P) y *signal* (V). Dependiendo de la definición exacta del semáforo, la operación P puede producir el bloqueo de un proceso, y la operación V puede producir que un proceso bloqueado deje de estarlo [Stallings, 97].

que lleguen mientras L tenga reservado el recurso. Los puntos suspensivos en H y M representan que el tiempo de bloqueo de la tarea H es impredecible. Finalmente, M termina en t_4 y L prosigue su ejecución, liberando el recurso en t_5. En ese punto termina la inversión de prioridades, ya que L es removida del procesador para permitir la ejecución de H.

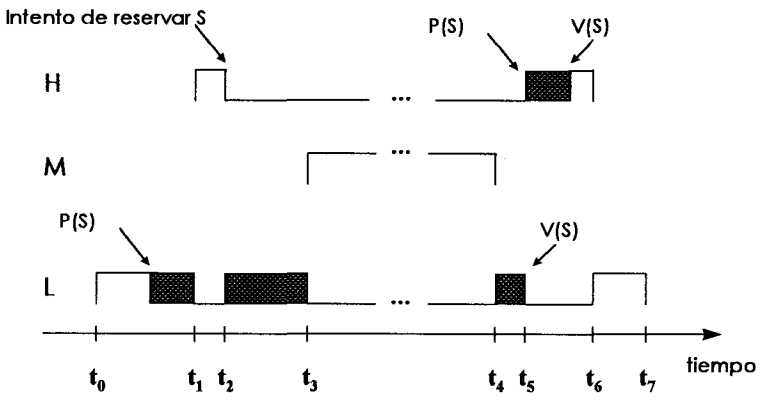


Figura 2.7 Inversión de prioridades.

2.2.2 Cooperación entre tareas

La cooperación entre tareas puede ser por comunicación o por compartición de datos [Stallings, 97].

Cooperación por comunicación. En esta clase de cooperación todas las tareas participan en un esfuerzo común que las liga. La comunicación se da a través del intercambio de mensajes y constituye una manera de sincronizar o coordinar las actividades. Puesto que no hay nada compartido no se requiere control de exclusión mutua; sin embargo los problemas de interbloqueo e inanición siguen presentes.

Cooperación por compartición de datos. Varias tareas pueden acceder a variables, archivos o bases de datos compartidas. Las tareas pueden usar y actualizar dichos datos sin hacer referencia a otras tareas, pero saben que otras tareas tienen acceso a los mismos datos. Los problemas de exclusión mutua, interbloqueo, inanición e inversión de prioridades se conservan, con la diferencia de que los elementos de datos pueden ser accedidos de dos modos distintos, lectura y escritura, y sólo las operaciones de escritura deben ser mutuamente exclusivas.

2.3 Sistemas operativos de tiempo real

Un sistema operativo es una colección de elementos de *software*, *firmware*, e inclusive *hardware*, que controlan la ejecución de programas[2] y proveen el soporte para la actividad de asignación de recursos. El funcionamiento de un sistema operativo puede describirse como el de un administrador de recursos. Los recursos básicos de un sistema de cómputo son: procesadores, memoria principal, dispositivos periféricos y archivos de datos [Halang y Sacha, 92].

Un sistema operativo de tiempo real, difiere significativamente de los sistemas operativos de propósito general y de los sistemas operativos orientados a los negocios. Esta distinción se origina por las diferencias existentes en el ambiente de *hardware* de cada uno de ellos, así como por sus requerimientos de funciones y servicios del sistema. El requerimiento más importante de un sistema de tiempo real es que debe responder a eventos internos y externos dentro de plazos de respuesta específicos. Un evento es una condición que surge en un sistema computacional o en su ambiente, y que requiere algún procesamiento específico de algún programa de *software* apropiado [Halang y Sacha, 92].

2.3.1 Ambiente Multitarea

En un ambiente multitarea varios programas se ejecutan concurrentemente y además pueden compartir los recursos del sistema. Como se dijo previamente, esto propicia problemas de asignación de los recursos a las tareas y de protección de los recursos asignados a una tarea en particular.

Un ambiente multitarea puede implantarse de dos maneras: utilizando un sistema de múltiples procesadores ejecutando varias tareas diferentes al mismo tiempo, o dividiendo el tiempo del procesador entre todas las tareas que compiten por tiempo de ejecución. Dado que el número de tareas normalmente es mayor que el número de procesadores, no hay necesidad de distinguir entre estos dos casos, aún cuando se use un sistema de múltiples procesadores, ya que de todas formas debe distribuirse el tiempo del (los) procesador(es) entre las tareas [Halang y Sacha, 92].

---

[2] Hasta ahora se ha utilizado preferentemente el término tarea para describir un conjunto de instrucciones de software que son ejecutadas por un procesador, sin embargo, los términos programa y proceso pueden usarse como sustitutos.

El ciclo de vida de una tarea en un ambiente multitarea puede ser descrito a través de cuatro estados [Halang y Sacha, 92]:

- *Terminado*. Una tarea está en este estado antes de iniciar y después de terminar su ejecución. La tarea es registrada y colocada en la lista de tareas mantenida por el sistema operativo, pero no puede solicitar ni tener recursos asignados.
- *Listo*. La tarea está lista para ejecución y espera a ser ejecutada por el procesador. Esto significa que se cumplen todas las condiciones para iniciar y continuar su ejecución, y que todos los recursos solicitados por la tarea le han sido asignados, con excepción del procesador, el cual está ocupado ejecutando una de las otras tareas.
- *Bloqueado*. La tarea está esperando la asignación de un recurso, la ocurrencia de un evento o el término de un intervalo de tiempo especificado.
- *En ejecución*. El procesador ha sido asignado a la tarea y ésta se encuentra en ejecución.

La Figura 2.8 muestra los estados y las operaciones para la transición de estados.

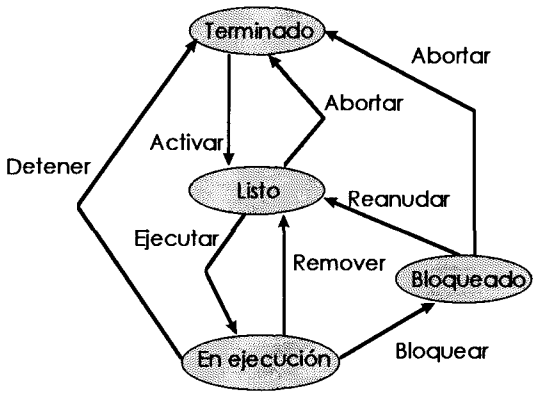


Figura 2.8 Diagrama de transición de estados de una tarea [Halang y Sacha, 92].

En un sistema de un solo procesador varias tareas pueden estar listas, pero sólo una de ellas puede estar en ejecución. Con base en un criterio ideal, un planificador[3] decide qué tarea será ejecutada, analizando los plazos de respuesta de las tareas que se encuentran en estado *Listo*.

Los cambios de estado de las tareas se producen debido a la ocurrencia de eventos dentro y fuera del sistema computacional. Los eventos pueden clasificarse en cuatro grupos [Halang y Sacha, 92]:

[3] Como se explicó en la sección 1.1, un planificador es un algoritmo cuya labor es hacer una programación de tareas que cumpla los plazos de respuesta de las tareas individuales.

- *Eventos Externos.* Corresponden a condiciones que se presentan en el ambiente computacional y son normalmente señaladas como interrupciones generadas por dispositivos periféricos.
- *Eventos de Tiempo.* Corresponden al término de intervalos de tiempo especificados. Estos eventos son indicados por el temporizador del sistema por medio de interrupciones de hardware.
- *Eventos Internos.* Corresponden a errores y excepciones que surgen en el sistema computacional durante la ejecución de las tareas. Estos eventos pueden indicarse a través de interrupciones de hardware o por rutinas de software.
- *Eventos de Programa.* Una tarea puede decidir hacer solicitudes de servicios en tiempo de ejecución[4], a través de la invocación directa o indirecta de interrupciones o llamadas a subprogramas del sistema operativo.

2.3.2 Elementos de un sistema operativo de tiempo real

Un sistema operativo de tiempo real multitarea consiste de un núcleo o *kernel* y de un conjunto de tareas del sistema *(shell)*. El *kernel* es responsable del manejo de los eventos y las tareas de aplicación, así como del control de las tareas del sistema. Estas últimas se encargan de proporcionar una variedad de servicios del sistema operativo (Figura 2.9) [Halang y Sacha, 92].

El *kernel* del sistema operativo detecta las interrupciones que señalan la ocurrencia de eventos a través de un monitor, el cual reconoce el tipo de interrupción y activa una rutina de manejo para el evento que la provocó (Figura 2.10). La reacción a un evento normalmente involucra una o varias transiciones de estado, por ejemplo: detener una tarea errónea o activar las tareas relacionadas a la ocurrencia de un evento externo. Finalmente, el planificador de tareas selecciona una tarea para ejecución y le entrega el control [Halang y Sacha, 92].

En el *kernel,* cada tarea se representa mediante una tabla de estado de la tarea, la cual contiene datos que describen su estado, prioridad y los recursos que le han sido asignados. El *kernel* cuenta también con un conjunto de subrutinas de transición de estado que implantan las operaciones *activar*,

[4] En general, por servicio en tiempo de ejecución debe entenderse una llamada al sistema operativo que puede ser invocada por las tareas del usuario [Milenković, 94].

abortar, *bloquear* y *reanudar*. Las operaciones *ejecutar* y *remover* son responsabilidad del planificador de tareas.

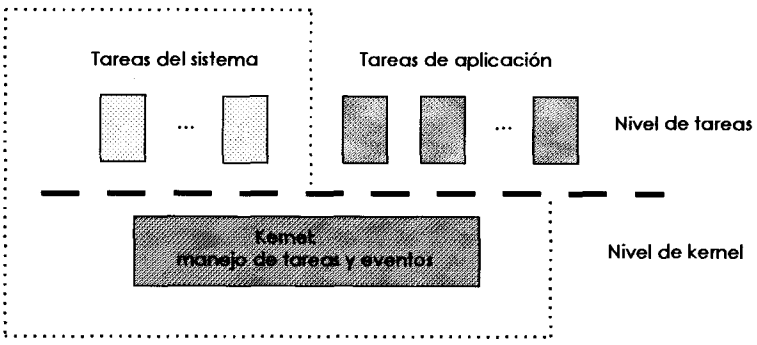


Figura 2.9 Modelo de un sistema operativo de tiempo real multitarea [Halang y Sacha, 92].

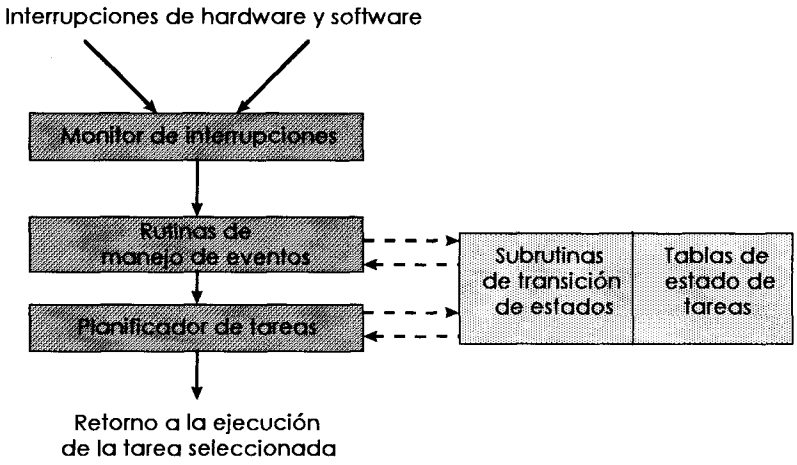


Figura 2.10 Modelo de un *kernel* multitarea [Halang y Sacha, 92].

2.4 Planificación en tiempo real

El planificador de tareas es el corazón de un sistema operativo de tiempo real, puesto que su labor es hacer una programación de tareas que cumpla las restricciones temporales de las tareas individuales. [Shin y Ramanathan, 94] describen el problema de planificación en un sistema de tiempo real de la siguiente manera: "*Dado el conjunto de tareas de un sistema de tiempo real y los recursos en el sistema, planificar es el proceso de decidir dónde y cuándo se ejecutará cada tarea.*"

A diferencia de otros sistemas en los que el objetivo de la planificación es minimizar el tiempo total requerido para ejecutar todas las tareas de la aplicación, un sistema de tiempo real busca cumplir las restricciones temporales de todas las tareas. Por ejemplo, en la Figura 2.11 se presenta una aplicación de tiempo real con las tareas *a, b, c, d, e* y *f* [Shin y Ramanathan, 94]. Las relaciones de precedencia se representan como se definió en la sección 2.1.2.3. Cada vértice tiene asociado un valor, el cual representa el tiempo requerido para efectuar el procesamiento de esa tarea. Las restricciones temporales de esta aplicación establecen que las tareas *e* y *f* deben terminar a más tardar en *31* y *16* unidades de tiempo respectivamente, asumiendo que todas las tareas tienen tiempo de liberación igual a cero.

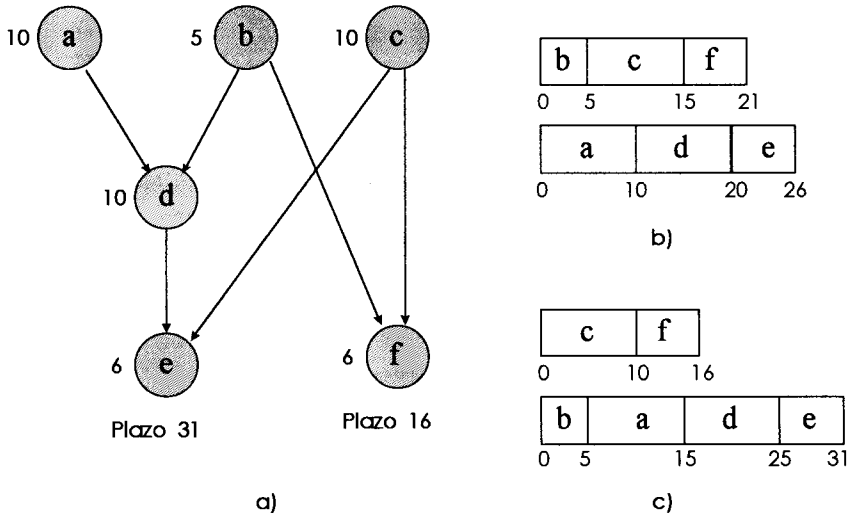


Figura 2.11 Diferencia entre los problemas de planificación de un sistema tradicional y un de sistema de tiempo real. (a) tareas de una sistema de tiempo real, (b) plan no válido para un sistema de tiempo real y (c) plan válido para un sistema de tiempo real [Shin y Ramanathan, 94].

En las Figuras 2.11(b) y (c) se presentan dos planes para la ejecución de la aplicación en un sistema de dos procesadores. El primer plan (b) satisface las restricciones de precedencia y su longitud total en tiempo es menor que la del segundo plan (c). Sin embargo, la tarea *f* que tiene plazo de respuesta 16 termina en la unidad de tiempo 21, por lo que no cumple una de las restricciones temporales de la aplicación. Este plan es óptimo para una aplicación que busca minimizar el tiempo total de ejecución de la aplicación, pero no para una aplicación de tiempo real. El plan de la Figura 2.11(c) cumple las restricciones de precedencia y las restricciones temporales y aunque su longitud total es mayor, es un plan adecuado para la aplicación de tiempo real. Entonces, dada una aplicación como la de la Figura

2.11(a), el problema de planificación en tiempo real es el de identificar planes como el de Figura 2.11(c).

2.4. 1 Espacio del problema de planificación de tareas

Puesto que el problema de planificación de tareas de sistemas de tiempo real tiene varias dimensiones, no existe una taxonomía aceptada para describirlo [Stankovic et al., 95]. Sin embargo, podría definirse el espacio del problema como el que se presenta entre los siguientes extremos (Figura 2.12):

- Un sistema uniprocesador con tareas periódicas independientes.
- Un sistema distribuido con tareas periódicas y aperiódicas interdependientes experimentando sobrecargas.

El espacio del problema sugiere una posible división: planificación en sistemas uniprocesadores y sistemas multiprocesadores. El problema que ataca esta tesis se halla en el contexto de los sistemas uniprocesadores.

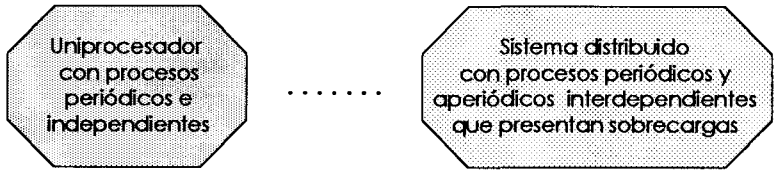


Figura 2.12 Espacio del problema de planificación de tareas de un sistema de tiempo real.

2.4.2 Métodos de planificación de sistemas de tiempo real.

En general, un método de planificación puede ser estático o dinámico. En las secciones siguientes se describen cada uno de estos métodos.

2.4.2.1 Planificación estática

La planificación estática se refiere al hecho de que el algoritmo de planificación conoce *a priori* todos los parámetros del conjunto de tareas que componen un sistema de tiempo real, tales como plazos de

respuesta, tiempos de procesamiento, restricciones de precedencia y tiempos de liberación futuros. En este tipo de algoritmos se genera un plan fijo simple o se asignan prioridades a las tareas de modo que pueda usarse un planificador de prioridades removible[5] convencional. Un ejemplo de algoritmos de planificación estática es el algoritmo de razón monotónica[6] (RMS, *rate monotonic scheduling*), que asigna prioridades estáticas a las tareas en función de sus períodos. Algunas veces hay confusión respecto a los tiempos de liberación futuros, pero si se conocen todos los tiempos de liberación futuros mientras el algoritmo está produciendo el plan, entonces todavía se puede hablar de planificación estática [Stankovic et al., 95].

2.4.2.2 Planificación dinámica

Un algoritmo de planificación dinámico es aquel que conoce completamente los parámetros de las tareas presentes actualmente en el sistema, pero desconoce los parámetros de otras tareas que también deben ser planificadas [Stankovic et al., 95].

La planificación dinámica es muy importante para los sistemas de tiempo real porque una de sus funciones principales es el manejo de eventos externos, los cuales normalmente son impredecibles en su ocurrencia. Un evento puede ocurrir en cualquier momento y los parámetros de la tarea que debe atenderlo pueden ser arbitrarios. Desafortunadamente, la naturaleza impredecible de los eventos externos hace que sea imposible hacer un algoritmo de planificación dinámica óptimo [Shih y Liu, 92], por lo que se busca construir algoritmos heurísticos[7] que produzcan planes cercanos a los óptimos.

2.4.3 Planificación en sistemas uniprocesadores

La planificación en sistemas uniprocesadores puede dividirse en los casos en que se presentan bloqueos debido a la interacción entre tareas y aquellos que no los presentan [Burns, 1991].

---

[5] Un planificador removible (*preemptive*) es aquel que puede interrumpir la ejecución de una tarea y reanudarla posteriormente sin afectar su comportamiento. Generalmente, la remoción ocurre cuando una tarea de mayor prioridad está lista para ejecución [Burns, 91].

[6] Este algoritmo será explicado en la sección 2.4.4.

[7] Un algoritmo heurístico es aquel que implanta una técnica que permite obtener soluciones aproximadas para un problema para el cual no existen algoritmos exactos o para aquel en que los algoritmos exactos son computacionalmente intratables [Shapiro, 92].

Sistemas uniprocesadores sin bloqueos

Procesos periódicos independientes. Es el caso más simple y para él se ha mostrado que el algoritmo de razón monotónica es un esquema de planificación estática óptimo con costo de administración *(overhead)* bajo. Otro algoritmo también óptimo es el algoritmo monotónico de plazos (*deadline monotonic algorithm*) [Burns, 91].

Procesos aperiódicos independientes. Normalmente, los sistemas de tiempo real son una mezcla de procesos periódicos y aperiódicos. El algoritmo de primero el plazo más próximo (EDF, *earliest deadline first*) es un algoritmo de planificación óptimo cuando las tareas aperiódicas son esporádicas. Sin embargo, cuando las tareas aperiódicas no son esporádicas este esquema no funciona, ya que presenta inestabilidad durante las sobrecargas [Burns, 91].

Como alternativa se utilizan adaptaciones del algoritmo de razón monotónica, para que pueda trabajar con tareas aperiódicas, por ejemplo, utilizando un proceso servidor periódico cuya función es la de servir a uno o más procesos aperiódicos, asignándole a este proceso servidor el tiempo máximo de ejecución posible que aún permita a las tareas periódicas cumplir sus plazos de respuesta. La desventaja de este enfoque es que la capacidad del servidor podría ser incapaz de atender una ráfaga de tareas aperiódicas [Strosnider et al., 95].

Sistemas uniprocesadores con bloqueos

En las aplicaciones realistas las tareas interactúan para conseguir objetivos comunes, como se explicó en la sección 2.2. Desafortunadamente, el problema de decidir si es posible planificar tareas que comparten recursos o tienen control de concurrencia es NP-duro[8] [Burns, 91] [Stankovic et al., 95]. Esto no quiere decir que es imposible construir pruebas polinomiales de planificabilidad, sino que las pruebas necesarias y suficientes son NP-duras. Por ejemplo, un programa podría fallar una prueba de planificabilidad, pero ser planificable. Sin embargo, si pasa alguna prueba de planificabilidad se asegura que es planificable. Por ello, se hacen consideraciones sólo para aquellas interacciones entre procesos que permiten construir estas pruebas de planificabilidad, por ejemplo, la forma común de exclusión mutua.

[8] NP-duro es una clase de problemas para los que no existen algoritmos que los resuelvan en tiempo polinomial [Horowitz y Sahni, 78].

Las primitivas de control de concurrencia comúnmente usadas, como los semáforos, pueden producir fenómenos no deseados como el interbloqueo o la inversión de prioridades [Burns, 91].

Recordemos el ejemplo de la sección 2.2.1. Para minimizar el efecto de la inversión de prioridades pueden utilizarse varios enfoques [Burns, 91]:

- *Prohibir remoción*. La idea es prohibir la remoción de una tarea cuando está ejecutando una sección crítica. Esto previene que *M* se ejecute (en la Figura 2.7), pero de cualquier modo *H* se retrasa aunque no necesite el recurso que tiene *L*.

- *Prevención*. La inversión de prioridades puede ser evitada completamente si no se permite a un proceso acceder a una sección crítica cuando exista la posibilidad de que un proceso de mayor prioridad se bloquee. El principal problema es que este esquema introduce ociosidad en el sistema, lo cual reduce la utilización del procesador, en el peor de los casos hasta en 50% [Burns, 91].

- *Herencia de prioridades*. Es un protocolo que remueve la inversión de prioridades cambiando dinámicamente la prioridad de los procesos que causan el bloqueo. La/ idea básica es que cuando una tarea *L* bloquee una o más tareas de mayor prioridad, cambie su prioridad y ejecute su sección crítica al nivel de prioridad mayor de entre todas las tareas bloqueadas. En el ejemplo de la sección 2.2.1, una vez que *L* bloquea a *H*, la prioridad de *L* será elevada a *H* y como resultado *L* se ejecutará en preferencia al proceso intermedio *M*. Desafortunadamente, el protocolo de herencia de prioridades también tiene dos desventajas: a) no previene la ocurrencia de interbloqueos, y b) pueden formarse cadenas de bloqueos, ocasionando que una tarea de alta prioridad permanezca bloqueada durante mucho tiempo [Sha et al., 90]. Por ejemplo, supongamos que una tarea de prioridad baja *L* está en una sección crítica utilizando el recurso *1* y es removida del procesador por una tarea de prioridad media *M*, la cual inmediatamente entra en una sección crítica utilizando el recurso *2*. Si en ese momento llega una tarea de alta prioridad *H* que requiere acceder secuencialmente a los recursos *1* y *2*, *H* quedará bloqueada y se habrá formado una cadena de bloqueos. Como resultado, *H* estará bloqueada durante la ejecución de dos secciones críticas, la de *M* y la de *L*.

- Otros enfoques son el protocolo de techo de prioridad *(priority ceiling protocol)* y la emulación del protocolo de techo de prioridad *(ceiling priority protocol)* [Sha et al., 90] [Sha et al., 94]. Estos enfoques se describen a continuación.

Protocolo de techo de prioridad.

Es un protocolo que supera al de herencia de prioridades pues previene interbloqueos (del modelo de interbloqueo de solicitud de una sola unidad) y no permite la formación de cadenas de bloqueos. Además, este protocolo asegura que una tarea de alta prioridad se bloqueará cuando mucho una vez durante su ejecución [Sha et al., 90] [Burns, 91].

En este protocolo, todas las tareas tienen una prioridad estática, usualmente asignada por el algoritmo de razón monotónica. Cada tarea tiene también una prioridad dinámica que es el máximo entre su prioridad estática y la que hereda debido a que bloquea procesos de prioridad mayor.

La idea del protocolo es que cuando una tarea T remueva de ejecución a otra tarea que está ejecutando su sección crítica, y comience a ejecutar su propia sección crítica z, se garantice que la prioridad a la que se ejecutará z es mayor que las prioridades heredadas por todas las secciones críticas removidas. Si no se cumple esta condición, T es bloqueado sin que se le permita entrar a su sección crítica z y la tarea que bloquea a T hereda la prioridad de T. Asumiendo que el acceso a las secciones críticas está controlado por semáforos binarios, el comportamiento descrito se logra asignando un techo de prioridad a cada semáforo, el cual es igual a la prioridad de la tarea de prioridad más alta que podría usar dicho semáforo. El acceso a una sección crítica de la tarea T será posible si y sólo si su prioridad es mayor que los techos de prioridad de todos los semáforos actualmente activados por tareas diferentes a T [Sha et al., 90]. Siempre se permite la activación de un semáforo cuando no hay ningún otro semáforo activo. El efecto del protocolo es que se puede activar un segundo semáforo sólo si no existe un proceso de igual o mayor prioridad que use ambos semáforos.

La ventaja del protocolo de techo de prioridad es que un proceso de alta prioridad puede ser bloqueado solamente una vez por un proceso de menor prioridad, en cada activación. El costo de este resultado es que la mayoría de los procesos experimentarán el bloqueo.

Emulación del protocolo de techo de prioridad.

Puede obtenerse un comportamiento equivalente al del protocolo de techo de prioridad al elevar la prioridad del proceso al nivel del techo en cuanto se active el semáforo [Sha et al., 94].

Si un proceso L comparte una sección crítica con un proceso de alta prioridad H, entonces cuando se encuentre en la sección crítica se ejecutará con al menos la prioridad de H. Como resultado, no podrá ejecutarse algún otro proceso con prioridad menor a H (activando un segundo semáforo). Cuando H se haga ejecutable experimentará cuando mucho un bloqueo simple. Además, este bloqueo será en el inicio de su ciclo; H no será capaz de remover del CPU a L si L está ejecutándose con prioridad H. Una vez que el bloqueo inicial ha terminado, H correrá sin interferencia, excepto por la posible activación de un proceso de mayor prioridad.

2.4.4 Algoritmo de razón monotónica (RMS)

El algoritmo de razón monotónica es un algoritmo removible óptimo para la planificación de tareas periódicas independientes en un sistema uniprocesador. Aunque el esquema se propuso por primera vez en 1973 [Liu y Layland, 73], ha ganado mucha popularidad recientemente y sobre él se han construido extensiones que permiten trabajar con tareas aperiódicas [Ling y Tarng, 91] y recursos compartidos [Sha et al., 90].

El algoritmo de razón monotónica asigna prioridades a las tareas en función de sus períodos, asumiendo que el período y el plazo de respuesta son iguales, de modo que la tarea de prioridad más alta es la de período más corto, la segunda tarea de mayor prioridad es la del segundo período más corto y así sucesivamente. El efecto que se produce es que cuando hay más de una tarea lista para ejecución, primero se da servicio a la que tiene el período más corto [Stallings, 97].

Si una tarea periódica se ejecuta siempre hasta el final, es decir, si nunca se niega el servicio a la tarea por insuficiencia de recursos, la utilización del procesador por parte de la tarea es $U = \tau/\pi$, donde τ y π son el tiempo de ejecución en el peor de los casos y el período de la tarea, respectivamente.

Liu y Layland desarrollaron una prueba de planificabilidad para el algoritmo de razón monotónica, el cual se basa en comparar la utilización del procesador por parte del conjunto de tareas a planificar, contra un límite de planificabilidad. Ellos determinaron que un conjunto de tareas $\tau_1, \tau_2, ..., \tau_n$ con períodos y plazos iguales $\pi_1, \pi_2, ... \pi_n$ es planificable por RMS si

$$\tau_1/\pi_1 + \tau_2/\pi_2 + ... + \tau_n/\pi_n \leq n(2^{1/n} - 1) \quad (2.1)$$

Para una n grande el límite de planificabilidad (el miembro derecho de la ecuación 2.1) converge en *0.693*, por lo que mientras que todas las tareas produzcan una utilización global del procesador menor a ese valor, se garantiza que todas cumplirán su plazo de respuesta [Sha et al., 94] [Stankovic et al., 95].

2.4.5 Algoritmo de planificación de primero el plazo más próximo (EDF)

La idea es asignar la prioridad más alta a la tarea que tiene el plazo de respuesta más próximo, de modo que siempre se ejecute la tarea con el plazo de respuesta más cercano, minimizando la diferencia entre el tiempo de término de una tarea y su plazo de respuesta [Klein et al., 93]. Una vez que una tarea termina su ejecución, nuevamente se selecciona la tarea con el plazo más cercano y se realiza su ejecución.

EDF se comporta bien cuando las tareas son aperiódicas, pues el algoritmo hace lo mejor que pueden de acuerdo a la carga de trabajo actual. Cuando todos los plazos de respuesta pueden ser cumplidos, este algoritmo es equivalente a RMS. Sin embargo, la desventaja de EDF es que en presencia de una sobrecarga los plazos de respuesta se pierden de manera impredecible.

2.5 Resumen

Este capítulo presentó una revisión de varios conceptos importantes sobre sistemas de tiempo real. Aunque todas las secciones son importantes, una de las más relevantes para este trabajo es la sección 2.4, la cual se refiere a la de planificación de tareas en tiempo real. Un resultado importante de esta sección es que se identificó que el objetivo de la planificación en sistemas de tiempo real es cumplir las restricciones temporales de todas las tareas, y no minimizar el tiempo requerido para su ejecución, como se busca en la planificación de tareas en aplicaciones que no son de tiempo real. También en

esta sección se describieron los algoritmos RMS y EDF, los cuales son base de varias técnicas de planificación de tareas, incluyendo la desarrollada en este trabajo de tesis.

El siguiente capítulo presenta la computación flexible y los enfoques que se han desarrollado al respecto cuando el tiempo es un recurso restringido. Estos enfoques son de interés para los sistemas de tiempo real, ya que, para ellos, el recurso más valioso es el tiempo [Shin y Ramanathan, 94].

Capítulo 3

COMPUTACIÓN FLEXIBLE

Una vez introducidos los conceptos fundamentales sobre sistemas de tiempo real, se presenta en este capítulo el concepto de computación flexible y se explican los enfoques principales que se han desarrollado al respecto, como son la planificación *design-to-time*, los algoritmos *anytime* y la computación imprecisa. La sección 3.1 define lo que es la computación flexible y describe su origen. La sección 3.2 explica los algoritmos *anytime* y la forma en que se construyen aplicaciones complejas a partir de ellos. La sección 3.3 describe el enfoque de computación imprecisa, ahondando en los algoritmos de planificación de tareas que se han desarrollado usando esta técnica. La sección 3.4 explica la filosofía de la planificación *design-to-time*. La sección 3.5 presenta la relación existente entre los algoritmos *anytime* y la computación imprecisa y sugiere una notación genérica que será utilizada a lo largo de esta tesis. Finalmente, la sección 3.6 cita ejemplos de tareas y sistemas de tiempo real que pueden ser implantados con el enfoque de computación flexible.

3.1 Antecedentes

En los últimos años de la década de los ochentas surgió una nueva manera de ver la computación llamada computación flexible. Este enfoque se refiere a aquellos procedimientos de solución de problemas que permiten un compromiso paulatino entre la calidad de los resultados y la asignación

de recursos, tales como tiempo, memoria e información. Los sistemas que utilizan computación flexible adquieren la habilidad de adaptar la calidad de su respuesta a los cambios dinámicos en los requerimientos de precisión y a la incertidumbre o variación en la disponibilidad de los recursos [Horvitz, 97].

El término computación flexible fue utilizado por primera vez en 1987, por Horvitz, en un artículo sobre problemas de decisión en tiempos críticos [Horvitz, 87]. En dicho artículo Horvitz discute situaciones en que los recursos disponibles para realizar el análisis completo de un problema no son suficientes, considerando los costos y beneficios de aplicar procedimientos de aproximación para obtener resultados parciales con los recursos disponibles.

La computación flexible es una idea interesante para ser aplicada en los sistemas de tiempo real, especialmente sobre las estrategias de planificación de tareas. Cuando se habla de flexibilidad en la planificación de tareas de un sistema de tiempo real es importante distinguir dónde se localiza dicha flexibilidad. La podemos encontrar en los algoritmos de planificación, es decir, los algoritmos son capaces de modificar su comportamiento y adaptarse a las situaciones imperantes en el ambiente. También podemos encontrarla en las tareas, es decir, el algoritmo de planificación no cambia, lo que varía es el comportamiento de las tareas de acuerdo a la disponibilidad de recursos.

Dependiendo de dónde se ubica la flexibilidad, existen varios enfoques que buscan producir los mejores resultados posibles con el tiempo disponible:

- En las tareas:
  - ✓ Algoritmos *anytime*
  - ✓ Computación imprecisa
- En los algoritmos de planificación:
  - ✓ Planificación *design-to-time (design-to-time real-time scheduling)*

3.2 Algoritmos *anytime*

A finales de la década de los ochentas, más o menos al tiempo que nacía la computación flexible, Dean y Boddy acuñaron el término "algoritmos *anytime*" [Dean y Boddy, 88]. Aunque al principio eran solamente ideas relacionadas, actualmente podemos pensar que computación flexible es una idea

más amplia y que los algoritmos *anytime* son un método de implantar computación flexible cuando el recurso restringido es el tiempo.

Un algoritmo *anytime* es aquel que puede ser interrumpido en cualquier momento mientras trabaja sobre un problema, produciendo un resultado aproximado, el cual incrementa su calidad conforme se incrementa su tiempo de procesamiento. Existen varias métricas que permiten evaluar la calidad de los resultados [Grass, 96][Russell y Zilberstein, 91]:

- *Precisión*. Esta medida le indica al sistema que tan cerca de la respuesta correcta está el resultado producido. Por ejemplo, el resultado se acompaña con un rango de error.
- *Área de cobertura*. Esta medida indica al sistema qué tan profundamente se ha explorado un espacio potencial de búsqueda o una base de datos.
- *Certidumbre*. Esta medida establece qué tan correctos son los resultados devueltos por el algoritmo. Podría ser expresada como la probabilidad de que el resultado sea el resultado preciso.
- *Completez*. Esta medida devuelve el nivel de detalle del resultado. Por ejemplo, en un algoritmo de visión una medida de calidad de este tipo podría representar la resolución a la que se procesa una imagen. Puede iniciarse con una resolución muy baja, la cual se va mejorando con el paso del tiempo.

El perfil rendimiento es el componente más importante de un algoritmo *anytime*. Se encarga de estimar el valor esperado de las métricas recién descritas para un procedimiento específico, en función de su tiempo de ejecución [Russell y Zilberstein, 91]. La Tabla 3.1 y la Figura 3.1 muestran, con ejemplos, dos formas distintas en que pueden representarse los perfiles de rendimiento.

Tabla 3.1. Perfil de rendimiento estadístico.

1.00								8%	19%	24%	31%	37%
							15%	30%	17%	39%	33%	40%
					16%	10%	16%	25%	30%	22%	25%	23%
		8%	4%	17%	20%	22%	30%	24%	19%	15%	11%	
Calidad	9%	10%	23%	23%	23%	37%	31%	13%	15%			
	11%	14%	30%	17%	21%	18%	8%					
	22%	17%	25%	24%	15%	13%						
	40%	31%	15%	19%	5%							
	15%	20%	3%									
0.00	3%											
	0 s											**15 s**

Tiempo

La Tabla 3.1 muestra un ejemplo de un perfil de rendimiento, en el que los porcentajes en cada celda representan la probabilidad de que la calidad del resultado esté en determinado rango en un instante de tiempo dado (perfil de rendimiento estadístico).

La Figura 3.1 muestra un ejemplo de un perfil de rendimiento de trayectoria, en el que la calidad del resultado se representa gráficamente con respecto al tiempo.

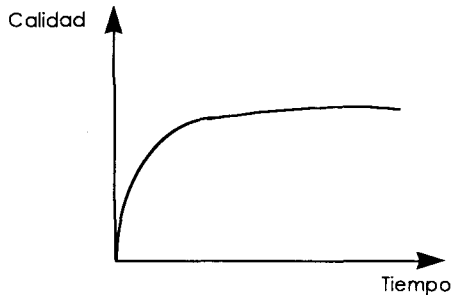


Figura 3.1 Perfil de rendimiento de trayectoria.

Un algoritmo *anytime* puede ser ejecutado de dos maneras [Russell y Zilberstein, 91]:

- Por contrato: asignándole un tiempo de ejecución fijo antes de ser invocado. El tiempo asignado depende del tiempo de procesamiento disponible. Si un algoritmo *anytime* por contrato es interrumpido anticipadamente, podría no producir resultados útiles.
- Interrumpible: proveyendo un método de interrupción para detener el algoritmo en cualquier momento y producir resultados de la calidad prevista en el perfil de rendimiento.

Composición de algoritmos *anytime*

Es raro que un sistema complejo se desarrolle con un solo algoritmo *anytime* que se ocupe de todas las actividades. Normalmente el sistema está formado por varios componentes dependientes unos de otros, los cuales se desarrollan y prueban de manera individual. Cuando dos o más de los componentes son algoritmos *anytime* es difícil hacer su composición, más aún si la salida de uno es la entrada del otro. Existen dos modelos de composición de los algoritmos *anytime*: lineal y de árbol (Figura 3.2) [Grass, 96]. En el modelo de composición lineal, la salida de un algoritmo *anytime* es la única entrada de otro algoritmo *anytime*, mientras que en el modelo de composición de árbol un algoritmo *anytime* puede tener varias entradas, que son a su vez la salida de varios algoritmos *anytime*.

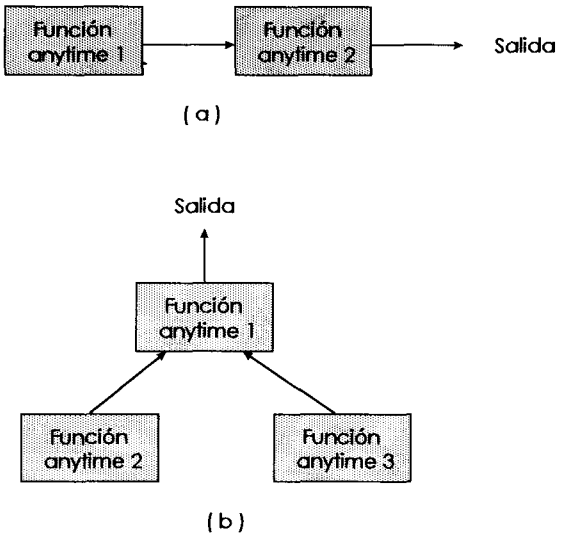


Figura 3.2 Modelos de composición (a) lineal y (b) de árbol.

Con objeto de que la composición sea óptima, los algoritmos *anytime* que reciban entrada de otros algoritmos *anytime* deben tener un perfil de rendimiento condicional. Un perfil de rendimiento condicional usa información acerca del tiempo y de la calidad de la entrada para determinar la calidad de la salida.

En ambos modelos de composición se selecciona el mejor tiempo de asignación buscando a través de todas las posibles asignaciones de tiempo para cada componente y evaluando la calidad esperada del resultado. Para ello, usando los perfiles de rendimiento se construye una función que reparta el tiempo de manera óptima entre las subfunciones de un algoritmo *anytime,* de modo que la calidad de la salida sea maximizada para el tiempo total asignado.

3.3 Computación imprecisa

La técnica de computación imprecisa es una manera de hacer que las tareas de un sistema de tiempo real no pierdan sus plazos de respuesta durante una sobrecarga, permitiendo una degradación paulatina del sistema. La idea es que cuando el sistema no disponga del tiempo necesario para producir un resultado exacto se produzca un resultado aproximado de calidad aceptable. Evidentemente esta técnica no es apropiada para cualquier aplicación, pues no en todos los casos los resultados imprecisos son útiles. La transferencia de imágenes de video y las operaciones de rastreo

(tracking) son ejemplos de aplicaciones en que es preferible un resultado impreciso a tiempo, que un resultado exacto demasiado tarde. Cualquier sistema que aplica la técnica de computación imprecisa se conoce como sistema impreciso. En un sistema de esta clase, cada tarea tiene una parte obligatoria y una parte opcional dependiente de la obligatoria. En condiciones normales, la parte opcional de una tarea se ejecuta totalmente obteniéndose un resultado exacto. Durante una sobrecarga, la parte opcional podría ejecutarse parcialmente produciéndose un resultado impreciso [Liu et al, 91][Liu et al., 94].

3.3.1 Modelo básico de computación imprecisa

El modelo básico de computación imprecisa [Liu et al., 94] caracteriza una aplicación como un conjunto de tareas $\boldsymbol{T}$. Cada tarea, T_i, es una tarea monótona (la calidad de su resultado siempre es creciente respecto al paso del tiempo) y está definida por los siguientes parámetros:

1. Tiempo de liberación *(ready time)* r_i. El instante de tiempo en el cual la tarea T_i está lista para ser ejecutada.
2. Plazo de respuesta *(deadline time)* d_i. El instante de tiempo en el que la tarea T_i ya debe haber sido terminada.
3. Tiempo de procesamiento τ_i. El tiempo necesario para ejecutar completamente la tarea T_i.
4. Peso w_i. Un número positivo que mide la importancia relativa de la tarea.

Cada tarea T_i se descompone lógicamente en dos partes o subtareas, la subtarea obligatoria M_i, con tiempo de procesamiento m_i y la subtarea opcional O_i, con tiempo de procesamiento o_i, de manera que $\tau_i = m_i + o_i$. Los tiempos de liberación y plazos de respuesta de las subtareas M_i y O_i son los mismos que para T_i. M_i es el predecesor inmediato de O_i.

Asumiendo un solo procesador, un plan es una asignación del procesador a las tareas de $\boldsymbol{T}$ en diferentes intervalos de tiempo. El tiempo de procesamiento asignado a una tarea es la suma de la longitud de los intervalos en los cuales el procesador está asignado a dicha tarea. En un plan válido, el procesador se asigna cuando mucho a una tarea a la vez, cada tarea se programa a ejecución después de su tiempo de liberación y después de que todos sus predecesores han terminado, y el tiempo de procesamiento asignado a la parte obligatoria M_i de T_i es igual al tiempo de procesamiento m_i. La subtarea opcional O_i puede terminar en cualquier momento. Cuando el tiempo de procesamiento α_i asignado a O_i es igual a o_i, se dice que T_i está planificada de manera precisa. Si

todas las tareas de un plan están planificadas de manera precisa, el plan es un plan preciso, pero si algún α_i es menor que o_i, el plan es impreciso. En los sistemas computacionales estándar sólo son aceptables los planes precisos, pero en la computación imprecisa ambos tipos de planes son útiles.

El error en el resultado producido por una tarea T_i es una manera de medir la calidad de sus resultados. En general, el error de una tarea representa la porción no ejecutada de su parte opcional. Por ejemplo, supóngase que el error se mide en un rango de 0 a 1. Un error nulo (error = 0) quiere decir que la parte opcional se ejecutó totalmente, de modo que el resultado producido por la tarea es un resultado preciso. Un error unitario (error = 1) significa que la parte opcional no fue siquiera ejecutada, por lo que el resultado producido (el generado por la parte obligatoria) es un resultado con la calidad mínima aceptable para ser un resultado útil. Cuando las tareas tienen relaciones de precedencia, el resultado producido por una tarea puede ser la entrada de sus sucesores. Si este resultado es impreciso, la entrada es errónea. El error en el resultado de una tarea T_i está en función de los errores de sus entradas e_i y de su tiempo de ejecución asignado σ_i, y se especifica por la función de error $\varepsilon_i(\sigma_i, e_i)$. Cuando las tareas son independientes, el error de las entradas puede ignorarse, por lo que el error de T_i es igual a $\varepsilon_i(\sigma_i)$.

El error en el resultado producido por una tarea puede modelarse de la siguiente manera (si las tareas son independientes):

- Si el tiempo σ_i asignado a una tarea es igual a su tiempo de procesamiento total τ_i, el error de su resultado es cero.
- Si el tiempo σ_i asignado a una tarea es igual al tiempo de procesamiento de su subtarea obligatoria m_i, el error de su resultado es igual a uno.
- Si el tiempo σ_i asignado a una tarea es menor que su tiempo de procesamiento τ_i, pero igual o mayor que su tiempo mínimo de procesamiento m_i, el error de su resultado $\varepsilon(\sigma_i)$ es una función decreciente de σ_i.

Generalmente, la función de error decrece en pasos discretos y por tanto es discontinua, pero puede aproximarse por una función continua, por ejemplo [Chung y Liu, 88]:

$$\varepsilon_i(\sigma_i) = 1, \qquad para\ 0 \le \sigma_i < m_i$$
$$\varepsilon_i(\sigma_i) = \left(1 - \frac{\sigma_i - m_i}{\tau_i - m_i}\right)^n, \ para\ m_i \le \sigma_i \le \tau_i \tag{3.1}$$

donde n es llamado el orden de la función de error. En general, los algoritmos de planificación garantizan la planificabilidad de la parte obligatoria, por lo que no es relevante el error en el intervalo $0 \le \sigma_i \le m_i$, sin embargo, para evitar discontinuidad en la función de error, se hizo $\varepsilon_i(\sigma_i) = 1$ para ese caso. Cuando n es mayor a 1, se dice que la función es convexa, esto quiere decir que el error disminuye más rápidamente en los primeras etapas de la subtarea opcional. Si n es igual a 1, la función de error es lineal, lo cual significa que el error decrece igualmente en todas las etapas de la subtarea opcional. Una función de error es cóncava cuando n es menor a 1, lo cual quiere decir que el error decrece poco en las primeras etapas de la subtarea opcional y mucho en las últimas etapas (Figura 3.3).

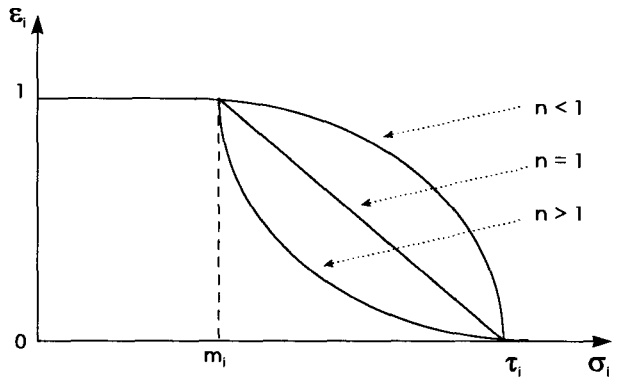


Figura 3.3 Función de error de tareas imprecisas [Chung y Liu, 88].

El error total ε del conjunto T de tareas es la suma de los errores de todas las tareas en T,

$$\varepsilon = \sum_{i=1}^{n} w_i \varepsilon_i(\sigma_i) \tag{3.2}$$

donde, $w_i > 0$ es el peso de cada tarea.

3.3.2 Modelo de tareas periódicas imprecisas *(imprecise-periodic-job model)*

El modelo de tareas periódicas imprecisas [Liu et al, 94] es una variación del modelo básico de computación imprecisa. La carga de trabajo a planificar es un conjunto $J = \{J_1, J_2, ..., J_n\}$ de n tareas periódicas independientes, donde una tarea periódica es una tarea que sufre una secuencia

periódica de activaciones. El tiempo de liberación y el plazo de respuesta de la tarea en cada período son el inicio y el fin del período, respectivamente. El período π_i de una tarea periódica J_i es la longitud del intervalo de tiempo entre dos activaciones consecutivas. El tiempo de procesamiento τ_i de la tarea J_i es el tiempo necesario para ejecutar completamente una activación de J_i. A veces es necesario referirse explícitamente a la activación de J_i en un período particular. $T_{i,j}$ es la activación de J_i en el *j-ésimo* período. J_i se compone de una subtarea obligatoria con tiempo de procesamiento m_i, y una subtarea opcional O_i, dependiente de la subtarea obligatoria, con tiempo de procesamiento $\tau_i - m_i$. El conjunto de las subtareas o partes obligatorias de las tareas de $\boldsymbol{J}$ se conoce como $\boldsymbol{M}$. $T_{i,j}(\boldsymbol{M})$ es la activación de M_i en el *j-ésimo* período y tiene el mismo tiempo de liberación y el mismo plazo de respuesta que $T_{i,j}$. El conjunto de las subtareas o partes opcionales de las tareas de $\boldsymbol{J}$ se conoce como $\boldsymbol{O}$. $T_{i,j}(\boldsymbol{O})$ es la activación de O_i en el *j-ésimo* período y tiene el mismo plazo de respuesta que $T_{i,j}$. El tiempo de liberación de $T_{i,j}(\boldsymbol{O})$ corresponde al tiempo en que la tarea $T_{i,j}(\boldsymbol{M})$ termina totalmente.

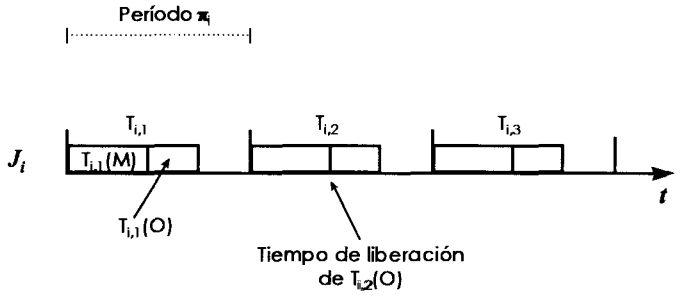


Figura 3.4 Ejecución de la tarea periódica imprecisa J_i

La Figura 3.4 muestra gráficamente algunos de los conceptos definidos en el párrafo anterior. Por simplicidad, el conjunto de tareas periódicas $\boldsymbol{J}$ está compuesto por una sola tarea periódica J_i.

Las funciones de error pueden ser modeladas de la misma forma que en el modelo básico de computación imprecisa, aunque es necesario reconsiderar la notación. Sea $\varepsilon_k(\sigma_{k,j})$ el error en el resultado producido por la tarea J_k en su *j-ésimo* período, cuando su tiempo asignado es $\sigma_{k,j}$. Entonces, la función de error $\varepsilon_k(\sigma_{k,j})$ está dada por [Chung y Liu, 88]:

$$\varepsilon_k(\sigma_{k,j}) = 1, \qquad para\ 0 \le \sigma_{k,j} < m_k$$
$$\varepsilon_k(\sigma_{k,j}) = \left(1 - \frac{\sigma_{k,j} - m_k}{\tau_k - m_k}\right)^n, \ para\ m_k \le \sigma_{k,j} \le \tau_k \tag{3.3}$$

3.3.3 Trabajos sobre planificación de tareas flexibles

Es en la Universidad de Illinois en Urbana Champaign donde se han desarrollado la mayoría de los trabajos sobre planificación utilizando computación imprecisa. Los problemas más importantes que se han estudiado en sistemas de un solo procesador y para los cuales se han generado algoritmos son [Liu et al., 94]:

- Planificación de tareas imprecisas monótonas minimizando el error total.
- Planificación de tareas periódicas minimizando el error promedio.
- Planificación para minimizar el error máximo. Dados un conjunto de tareas imprecisas y un plan de dichas tareas, el error máximo es igual al error de la tarea que tiene el error más grande cuando el conjunto de tareas se ejecuta de acuerdo al plan. Para resolver este problema se desarrollaron el *Slow Algorithm* y una variante del mismo [Shih y Liu, 95].
- Planificación con restricciones 0/1. En un plan que satisface la restricción 0/1, el tiempo de ejecución de cada tarea opcional es 0 u o_i. Las estrategias que se utilizan en la solución de este problema son la de primero la tarea con tiempo de procesamiento mayor *(Longest-Processing-time-First, LPF)*, que asegura que el error total del plan resultante será cuando mucho tres veces mayor al de un plan óptimo, y la de primero la tarea con tiempo de procesamiento menor *(Shortest-Processing-time-First, SPF)*, la cual asegura que el número de tareas opcionales no ejecutadas al usar esta estrategia es cuando mucho el doble las que no se ejecutarían usando un plan óptimo.

Los dos primeros problemas se describen detalladamente en las siguientes secciones, ya que un par de algoritmos desarrollados para esos casos son base del trabajo desarrollado en esta tesis.

3.3.3.1 Planificación de tareas imprecisas monótonas minimizando el error total

Todos los problemas de planificación imprecisa que han sido resueltos hasta la fecha son variaciones y extensiones del siguiente. Teniendo un conjunto de tareas monótonas y con pesos idénticos $T = \{T_1, T_2, \ldots, T_n\}$, en el que el error de cada tarea es igual al tiempo de la parte opcional no ejecutada, el problema es encontrar un plan removible viable de T que tenga el error total mínimo

entre todos los planes viables posibles (plan óptimo). Un plan viable es aquel plan válido en el que cada tarea termina dentro de su plazo de respuesta [Liu et al., 94].

Planificación estática

Cuando se conocen los parámetros de todas las tareas que serán planificadas antes del inicio de su ejecución, se dice que son tareas fuera de línea. Como se discutió en la sección 2.4.2.1, si un algoritmo de planificación maneja sólo tareas fuera de línea, se dice que es un algoritmo de planificación estático. Se han desarrollado dos algoritmos de planificación estática de tareas imprecisas, el algoritmo F y el algoritmo LWF.

Algoritmo F. El algoritmo F es un algoritmo estático óptimo para planificar tareas con pesos iguales (la misma prioridad relativa) que usa una versión modificada del algoritmo EDF clásico, llamada algoritmo ED [Liu et al., 91] [Liu, et al., 94].

Algoritmo de primero la tarea de mayor peso (Largest-Weight-First, LWF). Cuando los pesos de las tareas de T no son iguales, el algoritmo F no es óptimo, ya que no favorece la ejecución de las partes opcionales de las tareas de mayor peso. En este caso se utiliza el algoritmo LWF, el cual es un algoritmo estático óptimo para planificar tareas con pesos distintos [Liu et al., 91].

Planificación dinámica

Cuando las tareas a planificar llegan después de que alguna otra tarea ha iniciado su ejecución se denominan tareas en línea. Los parámetros de este tipo de tareas se conocen hasta el momento de su llegada. Si un algoritmo puede planificar tareas en línea se dice que es un algoritmo de planificación dinámico, según se discutió en la sección 2.4.2.2.

En el área de computación imprecisa, se han desarrollado algoritmos de planificación removibles dinámicos que minimizan el error total de todas las tareas [Shih y Liu, 92]. Shih y Liu demostraron que no se puede crear un algoritmo de planificación dinámico que siempre produzca un plan con error total mínimo, dado cualquier sistema compuesto de tareas flexibles en línea que tenga planes viables. Sin embargo, estos algoritmos son óptimos en el sentido de que producen planes viables con error total mínimo cuando todas las tareas cumplen la restricción de viabilidad de la parte obligatoria (*Mandatory Feasible Constraint:* MFC).

Un conjunto de tareas cumplen la restricción de viabilidad de la parte obligatoria si, en el instante de llegada de cada tarea en línea, su parte obligatoria puede ser planificada completamente junto con las partes obligatorias aún no terminadas de las tareas en línea que llegaron previamente, asegurando que ninguna pierda su plazo de respuesta [Liu et al., 94].

Se han considerado cuatro casos para el problema de planificación dinámica de tareas imprecisas [Shih y Liu, 92]:

a) No hay tareas fuera de línea y cada tarea en línea está lista para ejecución desde su llegada.
b) No hay tareas fuera de línea y las tareas en línea tienen tiempos de liberación arbitrarios.
c) Hay tareas fuera de línea y cada tarea en línea está lista para ejecución desde su llegada.
d) Hay tareas fuera de línea y las tareas en línea tienen tiempos de liberación arbitrarios.

Los casos b) y d) se reducen a uno solo si se considera que las tareas fuera de línea pueden manejarse como tareas en línea que ya llegaron, pero que aún no están listas para ejecución. Así pues, pueden considerarse solamente los casos a), c) y d).

Para el caso a) se ha desarrollado un algoritmo llamado NORA *(No-Off-line tasks and on-line tasks Ready upon Arrival)*, el cual será explicado detalladamente en la sección 4.3. Para el caso c) existe un algoritmo conocido como ORA *(On-line tasks Ready upon Arrival),* mientras que para el caso d), en que hay tareas fuera de línea y tareas en línea con tiempos de liberación arbitrarios, se desarrolló el algoritmo OAR *(On-line tasks with Arbitrary Ready time)* [Shih y Liu, 92].

Los tres algoritmos anteriores hacen uso de una lista de reservaciones, en la cual, para cada tarea aún no terminada, existen uno o más intervalos de tiempo reservados. Los intervalos de reservación se encuentran realizando una planificación inversa[1] *(reverse scheduling)* de la subtarea obligatoria M_i de la tarea T_i recién llegada, junto con las porciones aún no terminadas de las subtareas obligatorias de las tareas que arribaron previamente. Los intervalos de reservación encontrados se añaden a la lista de reservaciones.

La lista de reservaciones es una guía que permite a los algoritmos de planificación garantizar que las partes obligatorias de las tareas tengan tiempo suficiente para cumplir con sus plazos de respuesta.

[1] Ver sección 4.3.

De esta forma, los algoritmos no permiten que una subtarea opcional se ejecute durante un intervalo reservado. Conforme la subtarea M_i avanza en su ejecución se actualiza su reservación, de modo que su longitud corresponda a la porción de M_i que falta por ejecutar. Finalmente, cuando M_i termina, su reservación es cancelada totalmente.

Los algoritmos de planificación mantienen una cola en la que las tareas por ejecutar están ordenadas con base en la política del algoritmo EDF. La tarea del inicio de la cola ejecuta completamente su parte obligatoria y sigue con su parte opcional hasta que se alcanza el inicio de un intervalo reservado. En ese momento, la tarea se remueve de la cola, termina devolviendo su último resultado parcial, y la siguiente tarea comienza su ejecución.

Los algoritmos para los diferentes casos difieren en las estructuras de datos que utilizan para manipular la lista de reservaciones y la manera en que insertan, actualizan y cancelan los intervalos de reservación. En el caso más simple, el del algoritmo NORA, no es necesario saber qué intervalos están asignados a cada tarea, sólo se requiere saber su inicio y su término, así como cuanto tiempo hay reservado para cada una de las tareas. El caso más complejo es el del algoritmo OAR, en el cual se requiere saber específicamente la ubicación de los intervalos reservados para cada tarea [Liu et al, 94] [Shih y Liu, 92].

3.3.3.2 Planificación de tareas periódicas minimizando el error promedio

Dependiendo de los efectos causados por los errores, las aplicaciones se pueden dividir como aplicaciones de error no acumulativo (Tipo N) y aplicaciones de error acumulativo (Tipo C). Las aplicaciones Tipo N son aquellas para las que sus tareas periódicas, denominadas tareas períodicas Tipo N, son independientes y tienen entradas precisas en períodos distintos, y sólo es relevante el efecto promedio de sus errores. La transmisión de imágenes o video, el mejoramiento de calidad en imágenes y el procesamiento de voz son ejemplos de tareas de Tipo N. Las aplicaciones de Tipo C son aquellas en que los errores en diferentes períodos tienen efectos acumulativos, por lo que es necesario ejecutar totalmente la parte opcional de una activación de cada tarea periódica luego de varios períodos consecutivos para producir resultados precisos de vez en cuando. Las aplicaciones de control y telecontrol[2] son ejemplos de aplicaciones de Tipo C. [Chung et al., 90] [Liu et al., 94].

[2] Dirección de objetos a distancia.

Como un ejemplo de una aplicación Tipo C, considérese una tarea periódica en la que cada activación procesa una señal de un objeto que está siendo teledirigido y genera un comando para controlar el movimiento del objeto. Una activación que termina prematuramente produce estimados burdos sobre la posición, velocidad y aceleración del objeto, los cuales serán utilizados para producir el comando del control de movimiento. Si no se quiere perder el control sobre el objeto telecontrolado, es esencial que ocasionalmente se produzcan resultados precisos sobre su posición, velocidad y aceleración [Chung et al., 90].

3.3.3.2.1 Planificación de tareas periódicas de Tipo N

Para medir el rendimiento de los algoritmos de Tipo N, una buena medida es medir el error promedio de todos los resultados, de modo que lo que interesa es encontrar planes viables con el menor error promedio. Dado un plan viable del conjunto de tareas periódicas J, el error promedio se puede definir como sigue [Chung et al, 90]. Sea Q_k el número de períodos consecutivos sobre los cuales se calcula el error promedio de J_k. El error promedio de J_k, al final de su l-ésimo período es:

$$E_k = \frac{1}{Q_k} \sum_{j=l-Q_k+1}^{l} \varepsilon_k(\sigma_{k,j}) \tag{3.4}$$

para cualquier l mayor o igual a Q_k.

El error promedio de todas las tareas periódicas en J es:

$$E = \sum_{k=1}^{K} w_k E_k \tag{3.5}$$

donde w_k es un valor positivo constante que representa la importancia relativa de las diferentes tareas periódicas y K es la cardinalidad de J. Cuando todas las tareas periódicas son igualmente importantes, $w_k = 1 / K$.

Todos los algoritmos existentes para el caso de planificación de tareas periódicas imprecisas aseguran que se pueda cumplir la parte obligatoria de las tareas, permitiendo que las subtareas opcionales sean ejecutadas hasta que se hayan terminado todas las subtareas obligatorias que se encuentren listas para ejecución. Para lograrlo, los algoritmos son removibles y utilizan un esquema de manejo de prioridades en el que las subtareas opcionales tienen prioridades inferiores a las subtareas obligatorias. En general, las subtareas obligatorias se planifican de manera precisa con el algoritmo de razón monotónica. Ello se logra evaluando el límite de planificabilidad del algoritmo

de razón monotónica para el conjunto ***M*** de subtareas obligatorias, de modo que si la fracción de tiempo que mantienen el procesador ocupado es menor a $n(2^{1/n} - 1)$, con $n = K$, ***M*** es planificable de manera precisa [Liu et al., 94], [Chung, et al., 90].

Los algoritmos difieren en la manera en que se asigna la prioridad de las subtareas opcionales. Si las subtareas opcionales tienen la misma prioridad, se ordenan por antigüedad, de modo que primero se ejecute la tarea más antigua. Cuando se conoce el comportamiento de las funciones de error, existen tres algoritmos que pueden ser utilizados [Liu et al., 94], [Chung, et al., 90]:

Algoritmo de menor utilización (Least Utilization algorithm, LU). Este algoritmo tiene un buen rendimiento cuando la función de error es lineal. Las prioridades de las subtareas opcionales se asignan estáticamente de acuerdo con sus factores de utilización relativos $(\tau_k - m_k) / p_k w_k$, donde p_k es el plazo de respuesta de la tarea J_k en su primer período. Aquellas tareas periódicas que tengan factores de utilización más bajos tendrán prioridades más altas.

Algoritmo de menor tiempo de atención (Least Attained Time algorithm, LAT). Cuando las funciones de error son convexas, es decir, la reducción del error es mayor en las primeras etapas, este algoritmo tiene un mejor rendimiento. Las prioridades de las tareas se asignan de manera dinámica a las tareas opcionales. La estrategia de planificación de las subtareas opcionales es la siguiente: en cualquier momento en que el procesador no tenga subtareas de ***M*** por ejecutar, se asignará la prioridad mayor a aquella subtarea opcional que al momento haya sido atendida por menos tiempo.

Algoritmo de mejor retorno incremental (Best Incremental Return algorithm, BIR). Este algoritmo también asigna la prioridad de las subtareas opcionales de manera dinámica. Si se conoce de manera exacta el comportamiento de la función de error, en cualquier momento que el procesador no tenga subtareas de ***M*** por ejecutar se asigna la mayor prioridad a la subtarea opcional que pueda reducir el error de mejor manera. Para ello, debe calcularse, en cada rebanada de tiempo, la reducción incremental del error que puede lograr cada subtarea opcional presente en el sistema. Desafortunadamente, el cálculo dinámico de las reducciones incrementales del error, producen un costo de administración inaceptable en el sistema. Así pues, BIR se usa solamente como un parámetro útil en la comparación del rendimiento de los otros algoritmos.

Si no se conoce el comportamiento de la función de error, es preferible utilizar un algoritmo que intente minimizar el error promedio, asignando las prioridades más altas a las subtareas opcionales más urgentes. Se conocen tres algoritmos para desarrollar esa actividad [Liu et al., 94], [Chung, et al., 90]:

Algoritmo de menor longitud de período (Shortest Period Lenght algorithm, SPL). En este algoritmo las prioridades más altas se asignan a las subtareas opcionales cuyo período es más corto, de la misma manera que lo hace el algoritmo de razón monotónica.

Algoritmo de plazo más próximo (Earliest Deadline algorithm, ED). En este algoritmo la asignación de prioridades se hace dinámicamente. Las subtareas opcionales que reciben la prioridad más alta son aquellas que tienen el plazo de respuesta más próximo.

Algoritmo de menor tiempo de holgura (Least Slack Time algorithm, LST). Para cada subtarea opcional presente en el sistema se calcula su holgura, es decir, la diferencia entre su plazo de respuesta y el tiempo de ejecución que le falta, asignándole la prioridad mayor a aquella subtarea que tenga la holgura menor.

3.3.3.2.2 Planificación de tareas periódicas de Tipo C

Cuando una tarea periódica genera un error acumulativo es necesario que una activación de dicha tarea se ejecute completamente luego de un número dado de períodos consecutivos. Existen algoritmos basados en el algoritmo de razón monotónica que permiten planificar tareas periódicas de este tipo. Estos algoritmos deben considerar una subtarea opcional como si fuese obligatoria en algún momento, para lo cual utilizan alguna de las siguientes estrategias [Liu et al., 94]:

- *Primer período.* Se considera obligatoria la subtarea opcional de la primera activación de un número dado de períodos consecutivos.
- *Último período.* La parte opcional de una tarea periódica imprecisa se considera obligatoria en el último período de un número de períodos consecutivos determinado.
- *Período aleatorio.* En cada período, la parte opcional de una tarea periódica tiene una probabilidad constante de ser considerada obligatoria.
- *Aleatoria creciente.* La probabilidad de que una subtarea opcional sea considerada como obligatoria en cada período es mayor conforme va creciendo el error acumulativo.

3.4 Planificación *design-to-time*

Design-to-time es un enfoque útil para la solución de problemas de tiempo real en situaciones en que se tienen varios métodos para realizar cada una de las tareas que componen el problema. Es una alternativa a los enfoques que están basados en algoritmos *anytime* (e imprecisos como se verá en la sección 3.5). Su propósito es construir, en tiempo de ejecución, planes que combinen dinámicamente las técnicas de solución de las distintas partes del problema global, mientras maximizan la calidad de la solución [Garvey y Lesser, 96].

En otras palabras, este enfoque puede describirse como un método de solución de problemas en que, dado un período de tiempo, se construye y ejecuta dinámicamente un procedimiento de solución, el cual producirá una respuesta razonable utilizando todo el tiempo disponible. Los aspectos claves de este enfoque son: la disponibilidad de múltiples métodos que intercambian calidad por tiempo, el entendimiento de cómo interactúan y cómo pueden combinarse para resolver el problema global, considerando su duración y la calidad de sus resultados.

3.5 Relación entre algoritmos *anytime* y computación imprecisa

Aunque los enfoques de computación imprecisa y algoritmos *anytime* surgieron independientemente, se pueden apreciar algunas similitudes entre ellos. Ambos tratan con algoritmos que pueden ser interrumpidos y que establecen un compromiso entre la calidad de sus resultados y su tiempo de ejecución. Cuando se requiere la construcción de sistemas complejos ambos proponen una manera de establecer dicho compromiso: la composición en el caso de los algoritmos *anytime*, y la planificación de tareas en el caso de la computación imprecisa. Además, las nociones de calidad y error son análogas. El error en el resultado de una tarea imprecisa T_i es una métrica relacionada con la métrica de completez utilizada en los algoritmos *anytime*.

Feng y Liu notaron una relación simple entre los algoritmos *anytime* y la computación imprecisa: una subtarea opcional en el enfoque de computación imprecisa es una tarea escrita como un algoritmo *anytime* interrumpible [Feng y Liu, 97]. Sin embargo, si se hace una observación detallada, puede considerarse que una tarea imprecisa está compuesta por dos algoritmos *anytime*. La subtarea obligatoria es un algoritmo *anytime* ejecutado por contrato, el cual de ser interrumpido

anticipadamente no garantiza ninguna calidad en el resultado. La subtarea opcional es un algoritmo *anytime* interrumpible, según la observación de Feng y Liu.

Como se aprecia en la discusión de los párrafos anteriores, los enfoques de computación imprecisa y algoritmos *anytime* son ideas relacionadas. Aunque el trabajo desarrollado en esta tesis se basa fundamentalmente en las ideas aportadas por el enfoque de computación imprecisa, en adelante no se hará distinción alguna entre uno y otro enfoque y los conceptos serán calificados con el adjetivo genérico flexible. Por ejemplo, los algoritmos *anytime* y las tareas imprecisas se denominarán tareas flexibles y el modelo básico de computación imprecisa se denominará modelo básico de computación flexible.

3.6 Aplicaciones flexibles

Al revisar la literatura disponible, se encontró que, en general, todos aquellos procesos que pueden realizarse progresivamente, pueden ser implantados como tareas flexibles. Algunos de los procesos de esta clase referidos en la literatura son [Chung y Liu, 89] [Hull et al., 95] [Feng y Liu, 97] [Ibargüengoytia, 97]:

a) Operaciones de estimación estadística.
b) Problemas de planeación (encontrar el plan más corto para cumplir un objetivo, por ejemplo).
c) Transmisión y procesamiento progresivo de imágenes y video.
d) Procesamiento progresivo de voz.
e) Procesamiento progresivo de consultas.
f) Rastreo de objetivos utilizando radares.
g) Razonamiento probabilístico.

Además de los ejemplo mencionados, cuando hay varios métodos para efectuar un proceso, con diferentes tiempos de ejecución y con resultados de mayor o menor calidad para cada uno, dicho proceso también puede ser implantado como una tarea flexible. La idea es que exista una parte obligatoria que produzca un resultado de calidad aceptable utilizando alguno de los métodos disponibles y una parte opcional que proporcione una segunda opinión o refine este resultado a través de la utilización de uno o varios de los métodos restantes. Por ejemplo, un proceso encargado de

evitar automáticamente colisiones entre varios vehículos en movimiento puede ser realizado usando uno o varios de los siguientes métodos [Davis et al., 95]:

a) Identificación y rastreo de vehículos a través de técnicas de visión.
b) Monitoreo de vehículos basado en la desviación de rayos infrarrojos.
c) Rastreo de vehículos a través de radares.
d) Comunicación entre vehículos.

Una tarea flexible que implante este proceso puede contener una parte obligatoria que implante alguno de los métodos descritos y una parte opcional que utilice al menos uno de los métodos restantes para refinar el resultado obtenido.

Aunque muy pocas, en la literatura también se encontraron referencias sobre algunos sistemas de tiempo real particulares que podrían estar compuestos por tareas flexibles. Por ejemplo, en [Cozman, 96] se menciona un sistema de telecontrol de robots móviles. En dicho sistema, un robot móvil en una misión espacial envía una serie de imágenes hacia la base de control de la misión. Una interfaz recibe las imágenes, las procesa (elimina el horizonte y busca estructuras montañosas) y las compara contra un mapa del ambiente en que se encuentra el robot para producir estimados sobre su posición. Las actividades de procesamiento de imágenes de este sistema requieren una capacidad de procesamiento importante por lo que son implantadas como tareas flexibles.

Otro ejemplo de un sistema de tiempo real compuesto por tareas flexibles es un sistema de control autónomo de vehículos [Davis et al., 95]. Un sistema de control autónomo de vehículos es un sistema que permite la operación de un vehículo sin intervención humana, buscando, además, mejorar la eficiencia en el consumo de combustible y reducir los tiempos de traslado. El sistema está constituido por un grupo de sensores y actuadores que son controlados por un conjunto de tareas periódicas y esporádicas flexibles que implantan operaciones para evitar colisiones con obstáculos u otros vehículos, seleccionar una ruta de viaje y controlar la posición del vehículo respecto al centro de un carril.

3.7 Resumen

Este capítulo introdujo las ideas básicas sobre computación flexible y describió los enfoques más importantes al respecto. Resultados importantes de este capítulo son la identificación de las similitudes entre los enfoques de computación imprecisa y los algoritmos *anytime*, así como la determinación de utilizar un lenguaje genérico para hacer referencia a ambos enfoques y al trabajo desarrollado en esta tesis. Por otro lado, aunque se identificaron los procesos que son susceptibles de ser implantados como tareas flexibles, se observó que existen pocas referencias sobre sistemas de tiempo real compuestos por tareas de este tipo.

Las nociones sobre sistemas de tiempo real y computación flexible introducidas en los capítulos 2 y 3 son fundamentales para comprender el método de Planificación de Tareas periódicas, aperiódicas y esporádicas Flexibles (PTF) desarrollado en este trabajo de tesis. El capítulo siguiente se encarga de describir dicho método.

Capítulo 4

PLANIFICACIÓN DE TAREAS FLEXIBLES

Los capítulos 2 y 3 reseñaron los conceptos fundamentales y trabajos existentes sobre sistemas de tiempo real y computación flexible que sirven como base para el desarrollo del método de planificación desarrollado en esta tesis, al cual llamaremos PTF (método de Planificación de Tareas periódicas, aperiódicas y esporádicas Flexibles). El método PTF es producto de la integración y adaptación de las ideas de dos algoritmos de planificación de tareas flexibles existentes: LAT (sección 3.3.3.2.1) y NORA (sección 3.3.3.1). El resultado es un método de planificación de tareas periódicas, aperiódicas y esporádicas flexibles, que permite conservar la estabilidad del sistema en la presencia de sobrecargas.

Este capítulo describe el desarrollo del método PTF, partiendo de la explicación y ejemplificación de los algoritmos base (LAT y NORA), hasta llegar a la unificación de éstas y otras ideas en un nuevo método de planificación de tareas flexibles.

La sección 4.1 hace una introducción sobre la motivación para desarrollar el método PTF y presenta el modelo computacional considerado y las suposiciones asumidas en el desarrollo. La sección 4.2 describe y ejemplifica el algoritmo LAT; una parte fundamental del método PTF. La sección 4.3

trata sobre el algoritmo NORA, del cual se extraen varias ideas para el diseño de PTF, explica el enfoque de planificación inversa y presenta un ejemplo completo de la operación del algoritmo NORA. La sección 4.4 presenta el método de planificación PTF. Para ello define las características y funcionalidad de un servidor periódico de actividad aperiódica (SPAA) y presenta una variante del algoritmo NORA que opera sobre dicho SPAA. La sección 4.5 describe la forma en que son tratadas las restricciones de recursos en el método PTF. En la sección 4.6 se desarrolla un análisis para garantizar la planificabilidad de un conjunto de tareas periódicas y un SPAA. En la sección 4.7 se desarrolla un análisis de la complejidad temporal, específicamente del costo de administración del método PTF. La sección 4.8 hace un resumen de las características necesarias en un sistema operativo de tiempo real que soporte el método PTF. Finalmente, la sección 4.9 presenta los trabajos de planificación de tareas flexibles relacionados con esta tesis.

4.1 Introducción

Al estudiar los trabajos desarrollados en el área de computación flexible, se encontró que se han desarrollado algoritmos para la planificación de tareas periódicas y para tareas que presentan actividad aperiódica [Chung et al., 90] [Shih y Liu, 92] [Liu et al., 94]. Sin embargo, estos métodos son independientes y no consideran la combinación de la actividad periódica, aperiódica y esporádica. Poco o nada se ha hecho sobre planificación de conjuntos de tareas periódicas, aperiódicas y esporádicas flexibles. En particular, sólo se encontró el trabajo de [Davis et al., 95], el cual se comenta en la sección 4.9. En la práctica, un sistema de tiempo real normalmente se compone de una combinación de tareas periódicas, aperiódicas y esporádicas que cooperan de algún modo en la consecución de un objetivo común [Strosnider et al., 95]. Si consideramos que todas o algunas de dichas tareas pueden ser flexibles, resulta de sumo interés contar con un método que permita la planificación de sistemas de tiempo real de este tipo.

En esta tesis se desarrolla un método de planificación de tareas periódicas, esporádicas y aperiódicas construidas como algoritmos flexibles, apoyado en los trabajos previos de [Chung et al., 90] [Shih y Liu, 92] que permite mantener la estabilidad del sistema en presencia de sobrecargas, minimizando el error promedio de las tareas periódicas y el error total de las tareas aperiódicas y esporádicas.

Modelo computacional

Se consideró un modelo computacional simple que consiste de un conjunto de tareas periódicas que respetan el modelo de tareas periódicas flexibles [Liu et al., 94] y un conjunto de tareas aperiódicas y esporádicas que respetan el modelo básico de computación flexible [Liu et al., 94]. Ambos modelos fueron descritos en el capítulo 3. Otras suposiciones son las siguientes:

1. Las tareas no presentan restricciones de precedencia.
2. Todas las tareas son monótonas.
3. Las tareas periódicas flexibles tienen la misma importancia relativa y plazo de respuesta estricto, por lo que debe garantizarse su planificabilidad (son tareas críticas).
4. Las tareas aperiódicas y esporádicas flexibles tienen la misma importancia relativa entre sí y plazos de respuesta flexibles (ver sección 4.3).
5. Las tareas periódicas pueden interactuar a través de la compartición de recursos, por lo que debe asegurarse la exclusión mutua.
6. Las secciones críticas de las tareas periódicas están anidadas apropiadamente.
7. Los tiempos de cambio de contexto se consideran despreciables.

4.2 Planificación de las tareas periódicas

El problema de asegurar que al menos la parte obligatoria de las tareas periódicas sea ejecutada totalmente en presencia de una sobrecarga fue estudiado por [Chung et al., 90]. Ellos propusieron varios algoritmos para resolver dicho problema, los cuales varían dependiendo si las tareas periódicas son de Tipo N o de Tipo C, y de la función de error de las tareas flexibles.

Actualmente en el ITESM, Campus Morelos se desarrollan trabajos sobre transmisión de imágenes de video [Canedo, 97] y reconocimiento de voz [Uraga, 98]. Puesto que las tareas periódicas relacionadas a estas actividades son de Tipo N, se decidió enfocar este trabajo hacia ese tipo de tareas, pensando en desarrollar futuras aplicaciones de tiempo real que combinen ambos trabajos.

La función de error de una tarea flexible puede ser convexa, lineal o cóncava. El primer caso resulta el más interesante, ya que, al parecer, la mayoría de las tareas flexibles se comportan de ese modo. Como se discutió en el capítulo anterior, cuando las tareas son de Tipo N y la función de error de las tareas es convexa, el algoritmo LAT es la mejor opción para la planificación de tareas periódicas

flexibles. Con LAT se pueden asegurar los plazos de respuesta de las tareas periódicas, sin pensar aún en la presencia de tareas aperiódicas y esporádicas.

A continuación se muestran los planes producidos por RMS y LAT para conjuntos similares de tareas, con objeto de mostrar los beneficios de este último algoritmo. El ejemplo de la Figura 4.1 muestra el plan producido por el algoritmo de razón monotónica para el conjunto de tareas $J = \{ J_1, J_2, J_3, J_4 \}$ descrito en la Tabla 4.1.

Tabla 4.1 Conjunto de tareas periódicas del ejemplo de la Figura 4.1.

Tarea	*Período (π_i)*	*Tiempo de procesamiento (τ_i)*
J_1	20	10
J_2	40	5
J_3	50	5
J_4	60	15

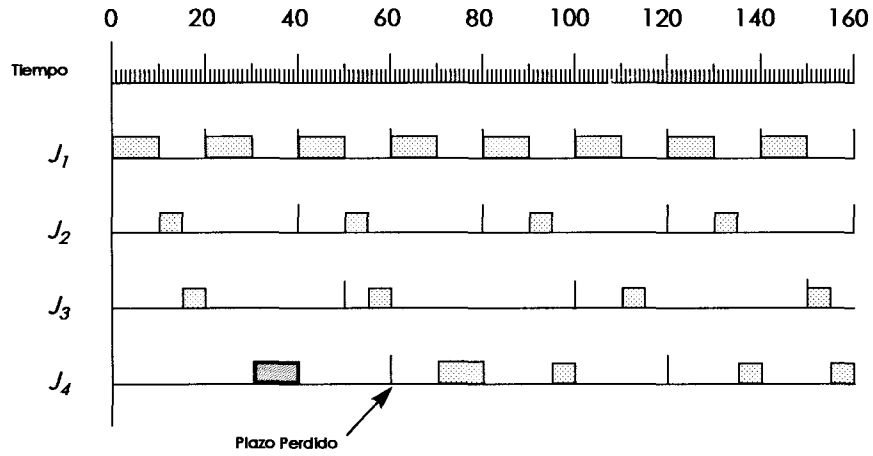


Figura 4.1 Plan de J usando RMS

En la Figura 4.1 se puede apreciar que J no es planificable con el algoritmo de razón monotónica, ya que la primera activación de J_4 pierde su plazo de respuesta en el tiempo *60,* habiéndose ejecutado sólo *10* de sus *15* unidades de tiempo de procesamiento. Sin embargo, si cada tarea periódica en J es una tarea flexible, LAT permite encontrar un plan flexible que minimiza el error promedio de las tareas periódicas.

La Figura 4.2 muestra el plan producido por LAT para el conjunto de tareas $J = \{ J_1, J_2, J_3, J_4 \}$ descrito por la Tabla 4.2.

Tabla 4.2 Conjunto de tareas periódicas para el ejemplo de la Figura 4.2.

Tarea	*Período* (π_i)	*Tiempo de procesamiento* (τ_i)	*Parte obligatoria* (m_i)	*Parte opcional* (o_i)
J_1	20	10	5	5
J_2	40	5	2	3
J_3	50	5	1	4
J_4	60	15	10	5

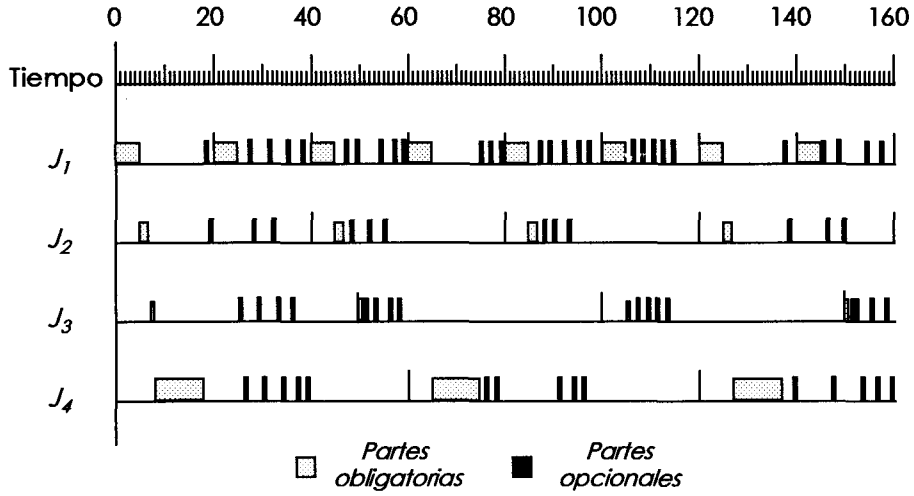


Figura 4.2 Plan de ***J*** usando LAT

Para el conjunto obligatorio ***M***, compuesto por las subtareas obligatorias de ***J***, la utilización total del procesador es *0.4867*. Este conjunto es planificable de acuerdo con RMS, puesto que $0.4867 \leq 0.7568$, el valor de utilización límite para garantizar la planificabilidad de un conjunto de tareas mediante el algoritmo de razón monotónica, calculado con la fórmula $n(2^{1/n} - 1)$, para $n = 4$. Puesto que el procesador está ocupado solamente el *48.67 %* del tiempo ejecutando las subtareas obligatorias del conjunto de tareas periódicas, la fracción de tiempo restante está disponible para ejecutar tareas del conjunto opcional ***O***, compuesto por todas las subtareas opcionales de ***J***.

En la Figura 4.2 también se aprecia que el algoritmo LAT produce una gran cantidad de cambios de contexto al planificar las partes opcionales de las tareas. Este es el costo por buscar minimizar el error promedio. En casos en que los tiempos necesarios para hacer cambios de contexto sean significativos, el uso de LAT podría provocar una disminución drástica del rendimiento del sistema. En estas circunstancias valdría la pena evaluar el rendimiento de algoritmos que asignen estáticamente la prioridad de las subtareas opcionales, como lo hace LU (ver sección 3.3.3.2.1).

Desafortunadamente, LU se comporta mejor para funciones de error lineales que para funciones de error convexas [Chung et al., 90].

4.3 Planificación de las tareas aperiódicas y esporádicas

En el capítulo 2 se estableció la diferencia entre las tareas aperiódicas y esporádicas. En resumen:

- Las tareas aperiódicas tienen plazos de respuesta flexibles y se caracterizan por tener un esquema de activación estocástico (por ejemplo, Poisson).
- Las tareas esporádicas son casos especiales de tareas aperiódicas que pueden tener plazos de respuesta estrictos y que se caracterizan por una separación mínima entre dos sucesos de activación consecutivos.

Uno de los objetivos de la computación flexible es evitar que las tareas pierdan sus plazos de respuesta [Liu et al., 94]. Por otro lado, una tarea con plazo flexible (como podría ser una tarea aperiódica) reduce su utilidad al perder su plazo. Si una tarea flexible con plazo flexible pierde su plazo y además produce un resultado aproximado, su utilidad podría no ser la esperada de acuerdo a una función de utilidad como la descrita por la Figura 2.3. Entonces, debiera establecerse una nueva y más compleja noción de utilidad, la cual sería una función del tiempo de ejecución asignado a una tarea y el instante de tiempo en que ésta es terminada.

En el método desarrollado en esta tesis se respeta la idea de la computación flexible de buscar cumplir los plazos de respuesta de las tareas, por lo que a pesar de que en adelante se considerará que las tareas aperiódicas y esporádicas tienen plazos flexibles, debido a la aleatoreidad presente en sus esquemas de activación que afecta la predecibilidad del sistema, es importante que sean terminadas dentro de sus plazos de respuesta.

En adelante, el término 'aperiódica' se utilizará para hacer referencia a aquellas tareas que se caracterizan por tener un esquema de activación estocástico, y el término 'esporádica' para aquellas tareas en que existe un tiempo mínimo de separación entre dos activaciones consecutivas. En algunos casos, el término 'actividad aperiódica' se referirá de manera genérica tanto a tareas aperiódicas como a tareas esporádicas.

[Shih y Liu, 92] desarrollaron una serie de algoritmos para planificar tareas flexibles en línea, minimizando el error total del sistema. Como se describió en el capítulo anterior, las tareas en línea son aquellas que arriban al sistema una vez que otras tareas ya han iniciado su ejecución. Tal es el caso de las tareas aperiódicas y esporádicas. Por tanto, las tareas aperiódicas y esporádicas son tareas en línea.

Uno de los algoritmos de Shih y Liu, el algoritmo NORA, realiza la planificación de tareas en línea con plazos de respuesta y tiempos de procesamiento arbitrarios, produciendo planes en los que las tareas cumplen con su parte obligatoria y las partes opcionales son ejecutadas hasta donde es posible. Este algoritmo, al igual que los otros desarrollados para la planificación de tareas en línea [Shih y Liu, 92], es útil para tareas flexibles cuya función de error es lineal. Como se dijo en el capítulo anterior (sección 3.3.3.1), NORA asume que todas las tareas presentadas al planificador cumplen la restricción de viabilidad de la parte obligatoria (MFC).

La planificación de las tareas aperiódicas y esporádicas para el método desarrollado en esta tesis se realiza a través de una adaptación del algoritmo NORA, llevándolo a un marco periódico y combinándolo con el algoritmo LAT descrito en la sección previa. Esta combinación de estrategias permite producir planes de conjuntos de tareas periódicas, aperiódicas y esporádicas flexibles que permiten mantener al sistema estable en la presencia de sobrecargas.

Como se dijo anteriormente LAT es útil para tareas con funciones de error convexas, mientras que NORA es adecuado para tareas con funciones de error lineales. La adaptación del algoritmo NORA desarrollada para el método PTF es útil para tareas aperiódicas y esporádicas flexibles con funciones de error lineales. Lo ideal sería que el método PTF pudiera planificar tareas periódicas, aperiódicas y esporádicas cuyas funciones de error son todas convexas. Desafortunadamente, cuando se tienen funciones de error convexas para tareas en línea, determinar cual es la tarea que hace un mayor incremento en la calidad de su resultado puede producir costos de administración muy altos. En el desarrollo de esta tesis inicialmente se pensó en una adaptación al algoritmo NORA para tareas con funciones de error convexas. Sin embargo, se encontró que es necesario mantener un grupo adicional de estructuras de datos referentes a los requerimientos de procesamiento y al tiempo disponible (las cuales deben contener valores actualizados) para determinar qué tarea opcional ofrece un mejor incremento de calidad en un momento dado. Debido a que, mantener permanentemente

actualizadas estas estructuras impacta inevitablemente los costos de administración afectando la planificabilidad de las tareas, se decidió trabajar con tareas aperiódicas y esporádicas flexibles cuyas funciones de error son lineales.

Algoritmo NORA

La operación de NORA se basa en el uso de una lista de reservaciones, la cual se utiliza como guía para decidir donde planificar las subtareas opcionales y cuanto tiempo asignarles. La lista de reservaciones se obtiene a partir de un plan preciso viable de las subtareas obligatorias de todas las tareas que han llegado al sistema y que aún no han sido terminadas, el cual es producido por el enfoque de planificación inversa.

Planificación inversa

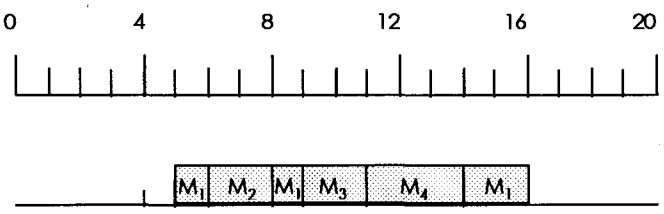
En este enfoque, las tareas se planifican en reversa, desde el tiempo que corresponde al plazo de respuesta más grande de las tareas presentes en el sistema, hasta el tiempo actual. Cada tarea se planifica en o antes de su plazo de respuesta, planificando primero aquellas tareas con tiempo de llegada más tardío. Un intervalo asignado a una tarea en el plan obtenido implica que en la lista de reservaciones se debe reservar un intervalo de tiempo para la ejecución de subtareas obligatorias. Es importante hacer notar que NORA no necesita saber cuales intervalos reservados corresponden a cada tarea, sólo requiere conocer donde inician y terminan los intervalos reservados [Shih y Liu, 92].

La lista de reservaciones está definida por los instantes de tiempo en que inician y terminan los intervalos reservados y la cantidad de tiempo reservada para cada subtarea obligatoria, de modo que la lista de reservaciones indica cuáles intervalos están reservados y cuáles no. Para asegurar que cada subtarea obligatoria cumpla con su plazo de respuesta, NORA evita planificar subtareas opcionales en un intervalo reservado.

La Figura 4.3 muestra un plan preciso viable construido mediante planificación inversa y la lista de reservaciones en el tiempo $t = 4$ para el conjunto de tareas descrito en la Tabla 4.3. Recordemos del capítulo 3, que para el caso del algoritmo NORA, el tiempo de liberación (r_i) de una tarea es igual a su tiempo de llegada, por ello el uso de la misma notación para ambos tiempos.

Tabla 4.3 Conjunto de tareas para el ejemplo de la Figura 4.3.

Tarea	*Tiempo de llegada (r_i)*	*Plazo (d_i)*	*Tiempo de ejecución de las partes obligatorias no terminadas (M_i).*
T_1	1	16	4
T_2	2	8	2
T_3	4	11	2
T_4	3	14	3



a) Plan producido por el algoritmo de planificación inversa

Tiempos reservados: $M_1 = 4$, $M_2 = 2$, $M_3 = 2$ y $M_4 = 3$

b) Lista de reservaciones

Figura 4.3 Ejemplo de lista de reservaciones y plan producidos por el algoritmo de planificación inversa. a) Plan producido, b) Lista de reservaciones.

Operación del algoritmo NORA

El planificador mantiene una cola de tareas ordenadas por prioridad. La prioridad está asignada con base en la política del algoritmo EDF. Al iniciar, la cola está vacía y el procesador está ocioso. Por cada arribo de una nueva tarea, el planificador hace una reservación en la lista de reservaciones, pone la nueva tarea en la cola y planifica para ejecución la tarea que está al inicio de la cola. La parte obligatoria de una tarea siempre se ejecuta antes que la opcional, la cual permanece en ejecución mientras haya tiempo disponible.

Durante el procesamiento de las tareas pueden ocurrir los siguientes eventos:

- *Evento 1:* La tarea actual T_i termina completamente o alcanza su plazo de respuesta.
- *Evento 2:* El tiempo actual t es el inicio de un intervalo reservado.
- *Evento 3:* Llegada de una nueva tarea.

El pseudocódigo de NORA muestra las decisiones de planificación que deben hacerse ante la ocurrencia de cada uno de los eventos ya descritos (ver Algoritmo 4.1).

Algoritmo 4.1 Pseudocódigo del algoritmo NORA.

1.- Mientras no ocurra algún evento, ejecutar la tarea que se encuentra al inicio de la cola de tareas.

2.- Si ocurre un evento, ejecuta el caso correspondiente:

2.1 *Evento 1:* La tarea actual T_i termina completamente o alcanza su plazo de respuesta.

- Eliminar la tarea actual de la cola de tareas.
- Cancelar la reservación de la tarea actual.
- Ir al paso 1.

2.2 *Evento 2:* El tiempo actual t es el inicio de un intervalo reservado.

- Si hay tiempo reservado para la tarea actual entonces
  cancelar la reservación de la tarea actual ;
  si no
  terminar la tarea actual y removerla de la cola de tareas.
- Ir al paso 1.

2.3 *Evento 3:* Llegada de una nueva tarea.

- Actualizar la reservación de la tarea actual.
- Hacer la reservación para la nueva tarea.
- Insertar la nueva tarea en la cola de tareas con base en la política del algoritmo EDF.
- Ir al paso 1.

El efectuar la cancelación o actualización de una reservación, o simplemente hacer una nueva, implica que la lista de reservaciones deja de ser válida. Para construir una nueva lista de reservaciones debe volverse a aplicar planificación inversa sobre el nuevo conjunto de tareas. Sin duda, esto toma mucho tiempo. Sin embargo, [Shih y Liu, 92] demostraron que no es necesario hacer una nueva planificación inversa, pues puede obtenerse el mismo efecto operando sobre la lista de reservaciones previa. Shih y Liu establecieron los siguientes lemas:

Lema 1. La nueva lista de reservaciones producida al insertar en la lista de reservaciones previa la subtarea obligatoria M_i de la tarea recién llegada T_i es la misma lista que se obtiene al usar el algoritmo de planificación inversa sobre todas las subtareas obligatorias aún no terminadas.

Lema 2. Si todas las tareas están listas, la reservación de x unidades de tiempo de la tarea con el plazo de respuesta más próximo puede cancelarse borrando del inicio de la lista de reservaciones los primeros intervalos reservados hasta alcanzar la longitud x.

Al ocurrir el Evento 1, debe liberarse la cantidad de tiempo x que aún tiene reservada la tarea T_i. Según el Lema 2, esto puede hacerse borrando los primeros intervalos de la lista de reservaciones hasta completar la longitud x. Adicionalmente T_i es eliminada de la cola y la primera tarea de la cola de tareas pasa a ejecución.

Cuando ocurre un Evento 2, es posible que en la lista de reservaciones aún exista algún tiempo reservado para la subtarea obligatoria actual M_i. En ese caso, debe cancelarse dicha reservación y continuar la ejecución de T_i. Sin embargo, si ya no hay tiempo reservado, simplemente se termina la tarea T_i y se pasa a ejecución la tarea que tiene el plazo de respuesta más próximo de entre todas las tareas aún no terminadas.

Con la ocurrencia de un Evento 3 debe actualizarse la reservación de la tarea actual T_i en la lista reservaciones. Si M_i requiere x unidades de tiempo de procesamiento extra para terminar totalmente, debe actualizarse su reservación añadiendo, o borrando, un intervalo reservado del inicio de la lista de reservaciones, de modo que el tiempo total reservado para M_i sea x. Luego, debe hacerse una reservación para la nueva tarea, lo cual, según el Lema 1, puede hacerse añadiendo a la lista de reservaciones uno o varios intervalos cuya longitud total sea igual a la subtarea obligatoria de la tarea recién llegada.

La Figura 4.4 muestra la lista de reservaciones y el plan producido por el algoritmo NORA para el conjunto de tareas descrito en la Tabla 4.4.

Tabla 4.4 Conjunto de tareas para el ejemplo de la Figura 4.4.

Tarea	***Tiempo de llegada (r_i)***	***Plazo (d_i)***	***Tiempo de ejecución (τ_i)***	***Parte obligatoria (m_i)***	***Parte opcional (o_i)***
T_1	1	6	6	3	3
T_2	2	14	7	5	2
T_3	10	20	6	4	2
T_4	9	16	6	4	2

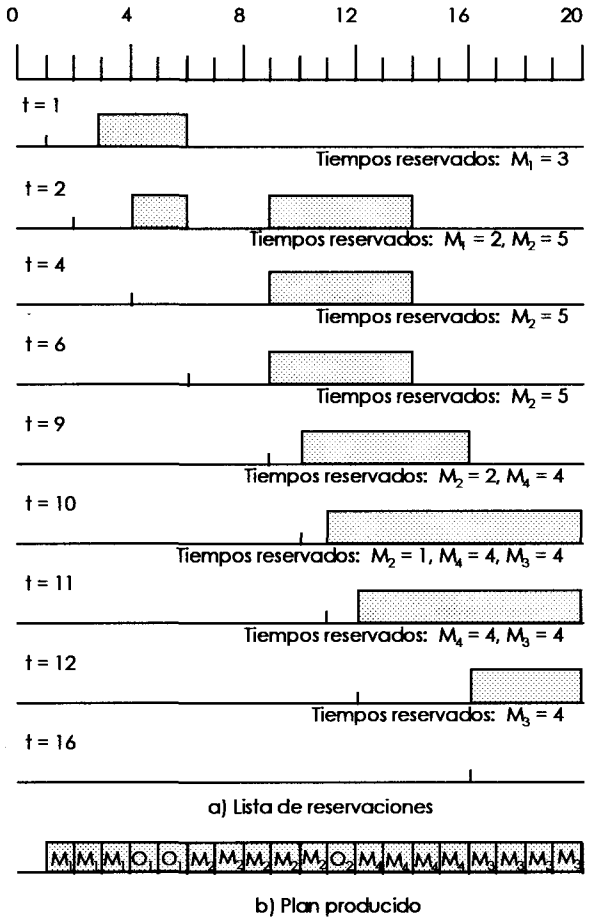


Figura 4.4 Ejemplo de ejecución del algoritmo NORA.

En el tiempo $t = 1$ llega la tarea T_1. El algoritmo le reserva el intervalo *(3, 6)* y como no hay ninguna otra tarea en el sistema, T_1 entra en ejecución. En $t = 2$ llega la tarea T_2. Se actualiza la reservación de T_1 y se reserva el intervalo *(9, 14)* para la tarea T_2. Como el plazo de respuesta de T_1 es menor al de T_2, T_1 es planificada para ejecución

En $t = 4$ se alcanza el inicio de un intervalo reservado, por lo que T_1, la cual aún tiene tiempo reservado, cancela su reservación. La tarea T_1 llega a su plazo de respuesta en $t = 6$, por lo que es eliminada de la cola de tareas y T_2 pasa a ejecución. En $t = 9$ se da la llegada de la tarea T_4. Se actualiza la reservación de T_2 y se reservan cuatro unidades de tiempo para la recién llegada. Similarmente, en $t = 10$ se actualiza la reservación de T_2 y se hace una reservación para la nueva tarea. En $t = 11$, se alcanza el inicio de un intervalo reservado. Como la tarea en ejecución T_2 aún tiene tiempo reservado, solamente cancela su reservación. El inicio de un intervalo reservado se alcanza nuevamente en $t = 12$, pero esta vez T_2 no tiene tiempo reservado, por lo que es eliminada del

sistema y T_4 pasa a ejecución, cancelando su propia reservación. Finalmente, en $t = 16$ se alcanza nuevamente el inicio de un intervalo reservado. Como T_4 ya no tiene tiempo reservado, es eliminada de la cola y T_3 entra en ejecución.

Si se quiere calcular el error total ε producido por el plan de la Figura 4.4(b), debe calcularse primero el error de cada tarea. Como este algoritmo es útil para tareas cuya función de error es lineal , puede hacerse uso de la Fórmula (3.1), con $n = 1$.

Del plan producido que se observa en la Figura 4.4(b), se tiene que $\sigma_1 = 5$, $\sigma_2 = 6$, $\sigma_3 = 4$ y $\sigma_4 = 4$. Usando la Fórmula 3.1 para T_1 se tiene que:

$$\varepsilon_1(\sigma_1) = \left(1 - \frac{5-3}{6-3}\right) = 0.333$$

De manera similar se obtienen $\varepsilon_2(\sigma_2) = 0.5$, $\varepsilon_3(\sigma_3) = 1.0$ y $\varepsilon_4(\sigma_4) = 1.0$. Considerando que todas las tareas tienen el mismo peso $w_i = 1$, partiendo de la Fórmula (3.2) se obtiene que el error total del conjunto de tareas es $\varepsilon = 2.833$. El error máximo posible es 4 (cuando todas las tareas tienen un error de 1), por lo que se produjo un error total del $70.83\ \%$. Este valor debe interpretarse como que en promedio, las tareas del sistema de tiempo real sólo lograron mejorar la calidad de sus resultados mínimos aceptables (los producidos por las subtareas obligatorias) en un 29.17 % respecto a los resultados precisos que se hubiesen producido si todas las tareas se hubiesen ejecutado totalmente (29.17 % = 100 % - 70.83 %).

4.4 Método de Planificación PTF

Para resolver el problema de planificación de tareas periódicas, aperiódicas y esporádicas flexibles, se aprovecharon las ideas de los algoritmos LAT y NORA discutidos previamente. El resultado es la integración de estas ideas en un método de planificación de tareas dinámico (método PTF) que permite conservar la estabilidad del sistema en la presencia de sobrecargas.

El problema de planificar tareas periódicas flexibles puede ser resuelto por el algoritmo LAT. Sin embargo, este algoritmo no es capaz de atender la actividad aperiódica y esporádica del sistema.

En sistemas de tiempo real convencionales (en el sentido de que no usan la técnica de computación flexible) se usan tres enfoques tradicionales para planificar una mezcla de tareas periódicas y aperiódicas [Lehoczky et al., 91] [Strosnider et al., 95]:

1. Permitir que las tareas aperiódicas interrumpan a las tareas periódicas y se ejecuten hasta su término.
2. Establecer un servidor de sondeo *(polling server)* que permita llevar la actividad aperiódica a un marco periódico, estableciendo periódicamente un tiempo de procesamiento para el uso exclusivo de las tareas aperiódicas.
3. Forzar a que las tareas aperiódicas reciban atención en trasfondo *(background)*, es decir, que sean ejecutadas cuando no haya tareas periódicas que ejecutar, lo cual se logra asignándoles un nivel de prioridad menor al de las tareas periódicas.

Un enfoque más avanzado es el uso de servidores de prioridad fija, como son, el servidor diferido *(deferreable server)* [Strosnider et al., 95] y el servidor esporádico *(sporadic server)* [Lehoczky et al., 91]. Estos esquemas son una mejora del servidor de sondeo, pues brindan una mayor flexibilidad, permitiendo que las tareas aperiódicas sean atendidas sobre demanda, en vez de en tiempos periódicos. Para ello, hacen que el tiempo de procesamiento exclusivo para las tareas aperiódicas esté disponible durante toda la longitud del período, o hasta que se agote.

4.4.1 Descripción del método.

La base del método PTF es la utilización de un servidor periódico sobre demanda (aprovechando las ideas de Lehoczky y Strosnider sobre servidores de prioridad fija), el cual es planificado junto con un conjunto de tareas periódicas flexibles mediante el algoritmo LAT. El servidor periódico es una tarea que proporciona una capacidad de procesamiento en cada período. Esta capacidad de procesamiento sirve para atender, sobre demanda, la ejecución de las tareas aperiódicas y esporádicas flexibles que lleguen al sistema, utilizando una variante del algoritmo NORA. Las tareas periódicas son insertadas en una lista de tareas listas para ejecución, mientras que, para el control de las tareas aperiódicas y esporádicas, existe una cola de actividad aperiódica, ordenada con base en la política del algoritmo EDF.

Las características del servidor periódico de actividad aperiódica (SPAA) utilizado son:

- El SPAA es una tarea periódica J_0 con período π_0 y capacidad de procesamiento τ_0. La determinación del valor adecuado de τ_0 es una labor complicada, por ello se discute a profundidad en la sección 4.6.2.
- Su período π_0 es el más corto de entre todas las tareas periódicas, y en consecuencia su prioridad es la más alta.
- El servidor opera en dos estados, activo o suspendido, dependiendo del estado actual de la cola de actividad aperiódica y de la capacidad de procesamiento disponible.
- El SPAA está en estado activo cuando hay alguna tarea en la cola de actividad aperiódica y su capacidad de procesamiento es mayor a cero. Si la capacidad de procesamiento está agotada, las tareas de la cola deberán esperar a ser atendidas en trasfondo o en el siguiente período del SPAA.
- El SPAA se encuentra en estado suspendido cuando la cola de actividad aperiódica está vacía o su capacidad de procesamiento está agotada, o ambas. A diferencia de un servidor de sondeo, el cual pierde su capacidad de procesamiento cuando no hay actividad aperiódica que atender, el SPAA conserva su capacidad de procesamiento restante durante todo su período. Al término de su período, el SPAA pierde la capacidad de procesamiento que no fue utilizada, y le es restablecida la capacidad de procesamiento total τ_0.
- Cuando el SPAA está suspendido, la llegada de una nueva tarea aperiódica o esporádica puede producir los siguientes efectos:
    - ✓ Si la capacidad de procesamiento está agotada, la tarea sólo es incluida en la cola de actividad aperiódica.
    - ✓ Si existe capacidad de procesamiento disponible y la tarea llega dentro del mismo período del SPAA en que éste fue suspendido, el SPAA pasa al estado activo y la tarea es atendida de inmediato.
    - ✓ Si existe capacidad de procesamiento disponible y la tarea llega en un tiempo t, que no pertenece al período del SPAA en que esté suspendido, el SPAA hace t el nuevo tiempo de inicio de su período, pasa al estado activo y la tarea es atendida de inmediato.

La política de atención de la actividad aperiódica en el servidor de tareas aperiódicas y esporádicas sobre demanda se basa en el algoritmo NORA, con las siguientes variantes:

1. El tiempo disponible para hacer reservaciones y ejecutar las tareas aperiódicas y esporádicas ya no es continuo, depende del período y capacidad de procesamiento del SPAA. Entonces, sólo podrá reservarse tiempo a las tareas aperiódicas y esporádicas en lista de reservaciones, dentro de los intervalos de tiempo que corresponden a los tiempos en que el SPAA se encontrará en ejecución.

Para ilustrar el manejo de la lista de reservaciones utilizando un SPAA, se considerará el ejemplo de la Figura 4.5. Supóngase que en el tiempo $t = 50$ se tiene un servidor periódico de actividad periódica J_0 con período $\pi_0 = 29$ y capacidad de procesamiento $\tau_0 = 6$, el cual ha agotado su capacidad en el período actual y tiene el inicio de sus siguientes dos períodos en los tiempos *60* y *89*. Además de un conjunto de tareas periódicas, se tienen las siguientes tareas aperiódicas (Tabla 4.5):

Tabla 4.5 Conjunto de tareas aperiódicas para el ejemplo de la Figura 4.5.

Tarea	***Tiempo de llegada (r_i)***	***Plazo (d_i)***	***Tiempo de ejecución de las partes obligatorias no terminadas (M_i).***
A_1	10	70	3
A_2	20	94	2

Supóngase que en el tiempo actual se da la llegada de la tarea aperiódica A_3, con $d_3 = 100$ y $m_3 = 5$. La Figura 4.5(a) muestra los intervalos reservados una vez que la tarea aperiódica del inicio de la cola (A_1) ha actualizado su reservación (los espacios en negro representan el tiempo que será utilizado por las tareas periódicas, ese tiempo no está disponible para hacer una reservación; los espacios en blanco representan el tiempo disponible para ser reservado y los espacios en gris, representan los intervalos reservados para A_1 y A_2). La Figura 4.5(b) muestra el estado de la lista de reservaciones una vez que se ha hecho la reservación para A_3. Para reservar las cinco unidades de tiempo necesarias, se tomaron los primeros *5* espacio disponibles en sentido inverso, a partir del tiempo *100*.

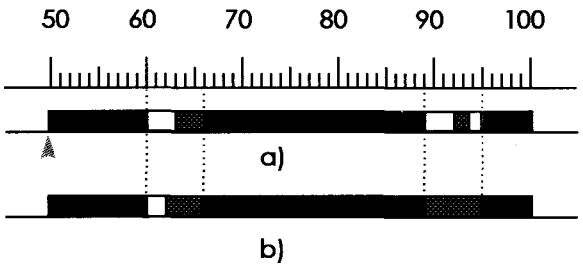


Figura 4.5 Reservación de una tarea en la lista de reservaciones usando el SPAA.
a) antes de la reservación de A_3, b) después de la reservación de A_3.

Como puede apreciarse, la operación de realizar una reservación en la lista de reservaciones sólo varía respecto a la del algoritmo NORA, en que únicamente debe considerarse la capacidad de procesamiento del SPAA. Esto también es válido para el resto de las operaciones sobre la lista de reservaciones.

2. La decisiones de planificación ante la ocurrencia de un Evento 2 se modifican de la siguiente manera con respecto al algoritmo NORA (del algoritmo 4.1):

Algoritmo 4.2 Modificación a las decisiones de planificación del algoritmo NORA.

Evento 2: El tiempo actual t es el inicio de un intervalo reservado.

- Si hay tiempo reservado para la tarea actual entonces
  cancelar la reservación de la tarea actual ;
  si no
  remover la tarea actual de la cola de tareas y ejecutarla en trasfondo con una prioridad más baja que la de cualquier subtarea periódica opcional.
- Ir al paso 1.

La modificación de las decisiones de planificación del Evento 2 permiten que una tarea que debiera ser eliminada tenga la oportunidad de incrementar la calidad de su resultado. Cuando hay varias tareas en trasfondo, éstas están ordenadas de acuerdo a su tiempo de llegada (por haber salido de una cola ordenada en base a EDF, el orden de llegada también corresponde al esquema EDF). La tareas en trasfondo son eliminadas y removidas de la lista de tareas listas para ejecución cuando cumplen su plazo de respuesta o cuando terminan totalmente su ejecución.

Además, para incrementar la utilización del procesador y reducir lo más posible el error de las tareas aperiódicas y esporádicas, cuando no haya tareas periódicas o aperiódicas/esporádicas en trasfondo listas para ejecución, el procesador ejecutará la tarea que esté al inicio de la cola de actividad aperiódica.

Aquellas tareas que no pasan la MFC no son presentadas al planificador y son tratadas por alguna operación de manejo de excepciones. La MFC, para una nueva tarea, consiste en verificar en la lista de reservaciones si hay suficiente tiempo disponible para hacer una reservación para la subtarea obligatoria de la nueva tarea.

El método de planificación descrito previamente conserva la estabilidad del sistema en las siguientes situaciones de sobrecarga:

- El tiempo disponible no es suficiente para cumplir los plazos de respuesta de las tareas periódicas completas. En este caso se ejecutan completamente las partes obligatorias de las tareas periódicas y luego se busca minimizar el error promedio en la ejecución de las partes opcionales.
- La actividad aperiódica rebasa la capacidad disponible en el servidor de actividad aperiódica. En este caso, sólo se aceptan las tareas aperiódicas y esporádicas cuya parte obligatoria sea planificable, se asegura la ejecución de la parte obligatoria de las tareas y se busca minimizar el error total en la ejecución de las partes opcionales.

4.4.2 Ejemplo

En la Figura 4.6 se muestra un plan producido por el método PTF para el conjunto de tareas mostrado en la Tabla 4.6. Las tareas cuya identificación inicia con *J*, *E* y *A*, son periódicas, esporádicas y aperiódicas respectivamente.

En este ejemplo (Figura 4.6) las puntas de flecha representan la llegada de una tarea aperiódica o esporádica. Las líneas verticales indican el término de un período y el inicio del siguiente, en el caso de las tareas periódicas y el plazo de respuesta, en el caso de las tareas aperiódicas y esporádicas.

Tabla 4.6. Conjunto de tareas del ejemplo de la Figura 4.6.

Tarea	*Período* (π_i)	*Plazo* (d_i)	*Tiempo de llegada* (r_i)	*Parte obligatoria* (m_i)	*Parte opcional* (o_i)
J_0	29	-	-	6	0
J_1	30	-	-	5	1
J_2	35	-	-	6	2
J_3	40	-	-	7	3
E_1	-	78	53	2	1
E_2	-	74	4	3	2
A_1	-	89	45	7	4
A_2	-	78	58	2	3
A_3	-	81	73	1	1

Vale la pena observar que la capacidad del servidor periódico no es utilizada completamente en el primer período, sólo se utilizan *5* de sus *6* unidades (en la ejecución de E_2), por lo que el resto se pierde en el tiempo $t = 29$. Como el SPAA es activado nuevamente hasta $t = 45$, en la llegada de la tarea A_1 y fuera del período en que fue suspendido, se traslada el inicio de su período hasta el tiempo $t = 45$, de modo que su siguiente período iniciara en $45 + \pi_0 = 45 + 29 = 74$.

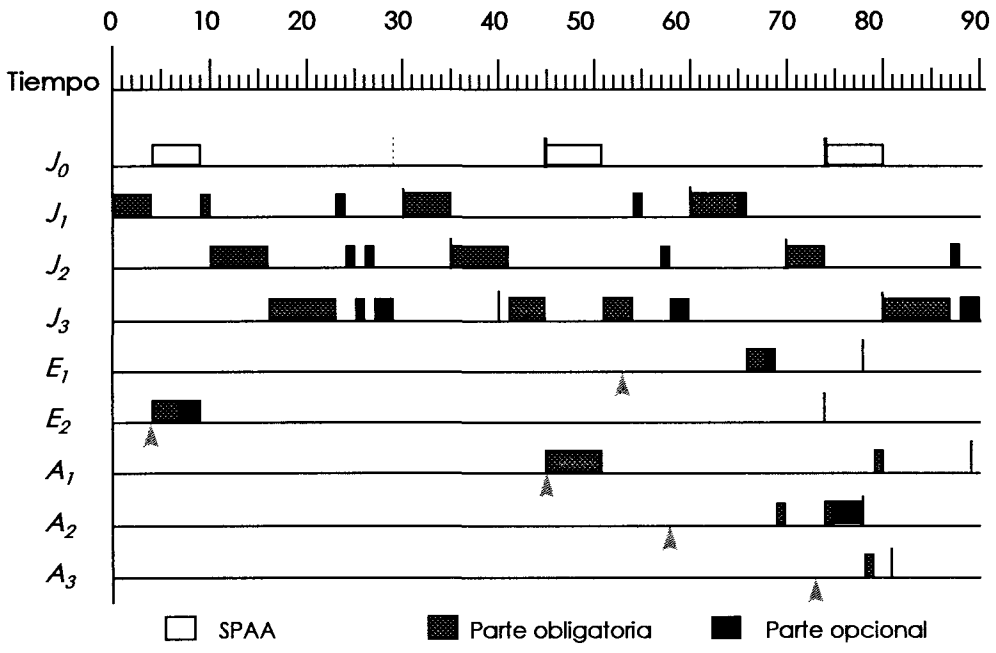


Figura 4.6 Ejemplo de ejecución del método de planificación de tareas periódicas, aperiódicas y esporádicas flexibles.

En el intervalo de tiempo *(66, 70)*, la capacidad de J_0 está agotada, sin embargo, puesto que el procesador no tiene actividad periódica que atender, aprovecha ese tiempo atendiendo las tareas que están al inicio de la cola de actividad aperiódica.

En la figura también se aprecia que las tareas A_1 y A_3 no ejecutan su subtarea opcional, sin embargo, logran completar su subtarea obligatoria antes de que se cumpla su plazo de respuesta.

4.5 Restricciones de recursos

En el capítulo 2 se hizo referencia al problema de decidir si es posible planificar un conjunto de tareas que comparten recursos, el cual es un problema NP-duro.

El método de planificación desarrollado tolera limitadamente el uso de recursos compartidos a través del protocolo de techo de prioridad, el cual previene, de una manera predecible, los problemas de inversión de prioridades y creación de interbloqueos [Sha et al., 90]. Las limitantes son que sólo las partes obligatorias de las tareas periódicas pueden utilizar recursos compartidos y sólo se considera el modelo de interbloqueo de solicitud de una sola unidad.

Permitir que las tareas aperiódicas, esporádicas o las subtareas opcionales de las tareas periódicas se apropien de recursos podría provocar situaciones de retrasos prolongados e impredecibles para las tareas periódicas. Por ejemplo, si una tarea aperiódica posee el recurso que necesita una tarea periódica, la política EDF para la atención de la actividad aperiódica podría provocar que la tarea aperiódica sea retrasada por varias tareas con plazos de respuesta más próximos, resultando en que el recurso es liberado luego de varios períodos del SPAA, cuando quizá la tarea periódica haya perdido su plazo. Un caso similar ocurre con las partes opcionales de las tareas periódicas. La política de privilegiar en todo momento a las subtareas opcionales que hayan tenido menor tiempo de procesamiento, podría propiciar que sea impredecible el tiempo que una tarea periódica espera un recurso que posee una subtarea periódica opcional.

Usar el protocolo de techo de prioridad requiere que se calculen los tiempos que una tarea podría estar bloqueada. Dicho tiempo se incluye en el análisis de planificabilidad de la sección 4.6.

4.6 Planificabilidad

En esta sección se desarrolla una prueba que garantiza la planificabilidad de un conjunto de tareas periódicas que utilizan recursos compartidos y un servidor periódico de actividad aperiódica, el cual actúa sobre demanda. Esta prueba de planificabilidad sirve como base para encontrar la capacidad de procesamiento τ_0 que puede tener un SPAA al ser planificado con un conjunto de tareas periódicas dado.

4.6.1 Análisis de Planificabilidad

[Lehoczky et al., 89] hicieron una caracterización exacta de la capacidad del algoritmo de razón monotónica para cumplir con los plazos de respuesta de un conjunto de tareas periódicas, que mejora los estudios realizados en 1973 por Liu y Layland. Los primeros determinaron que:

$$\max_{(1\le i\le n)} \min_{(t\in H_i)} \sum_{j=1}^{i} \tau_j \cdot \left\lceil \frac{t}{\pi_j} \right\rceil \le 1, \tag{4.1}$$

con $H_i = \{k \cdot \pi_j \mid j = 1, \ldots, i\ ;\ k = 1, \ldots, \lfloor \pi_i / \pi_j \rfloor\}$, es una condición necesaria y suficiente para garantizar la planificabilidad de un conjunto de tareas periódicas bajo las siguientes condiciones:

"Sea $J = \{J_1, J_2, \ldots, J_n\}$ un conjunto de n tareas periódicas, tales que J_i tiene período π_i, tiempo de ejecución τ_i y fase[1] relativa a 0, I_i, con $0 \le I_i \le \pi_i$. Esto quiere decir que las activaciones que corresponden a la tarea J_i inician en los tiempos $I_i + k\pi_i$, con $k \ge 0$. La activación que inicia en $I_i + k\pi_i$ tiene plazo de respuesta $I_i + (k+1)\pi_i$, es decir, el tiempo de inicio de la siguiente activación. Las tareas se etiquetan de modo que $\pi_1 \le \pi_2 \le \ldots \le \pi_n$, y se asigna a J_i la prioridad i. Se asume que las tareas están listas para ser ejecutadas desde su tiempo de inicio y que pueden ser removidas instantáneamente (ignorando cualquier clase de bloqueo)". H_i es el conjunto de instantes de decisión para la tarea J_i (en los que debe probarse la planificabilidad de J_i), y se compone del plazo de J_i y de los tiempos de inicio de todas las activaciones de las tareas periódicas de prioridad mayor a J_i que ocurren antes del plazo de J_i. La caracterización de [Lehoczky et al., 89] se basa en analizar la planificabilidad del conjunto de tareas en un instante crítico, es decir, cuando todas las tareas tienen el mismo tiempo de inicio, ya que Liu y Layland probaron que un conjunto de tareas periódicas es

---

[1] Fase es un desplazamiento en el tiempo relativo a algún evento o a un instante de tiempo específico [Klein et al., 93].

planificable por el algoritmo de razón monotónica si la primera activación de cada tarea cumple su plazo de respuesta al ser iniciada en un instante crítico [Liu y Layland, 73].

[Strosnider et al., 95] hicieron un análisis de planificabilidad para el caso en que se desea planificar un servidor periódico diferido J_o, con la prioridad más alta, tiempo de ejecución τ_0, período π_0 y fase relativa a 0 I_i junto con un conjunto de tareas periódicas $J = \{J_1, J_2, \ldots, J_n\}$. Este análisis será útil para determinar el tiempo de ejecución que se asignará al servidor periódico de actividad aperiódica del método PTF. El análisis se apoya en tres resultados obtenidos de los dos trabajos anteriores:

1. Una tarea J_i tiene su tiempo de respuesta más largo cuando inicia en un instante crítico. El tiempo de respuesta de una tarea se define como la diferencia entre el tiempo en el instante en que es terminada y su tiempo de inicio. Cuando $I_i = 0$, $1 \leq i \leq n$, ocurre un instante crítico en $t = 0$.
2. Usando el algoritmo de razón monotónica, todas las tareas cumplirán sus plazos de respuesta si la primera activación de cada tarea cumple su plazo de respuesta conforme a la fase de instante crítico *(critical instant phasing)* $I_i = 0$, $1 \leq i \leq n$.
3. Todos los plazos de respuesta de las tareas periódicas están garantizados por el algoritmo de razón monotónica, para todas las posibles fases de las tareas, si y sólo si se cumple la ecuación (4.1).

El objetivo del análisis de planificabilidad al incluir un SPAA es determinar las condiciones en que las tareas de J tienen su tiempo de respuesta más largo. Si en esas condiciones todas las tareas pueden cumplir sus plazos de respuesta, entonces el conjunto de tareas es planificable. Del análisis presentado en [Strosnider et al, 95] se obtiene que el tiempo de respuesta más largo para el conjunto de tareas se presenta cuando:

a) todas las tareas del conjunto tienen un mismo tiempo de inicio para alguna activación,
b) en ese mismo instante, el SPAA demanda τ_0 unidades de tiempo, y
c) el SPAA será reiniciado τ_0 unidades de tiempo, después.

Este caso se observa cuando el servidor periódico de actividad aperiódica es activado por la llegada de una tarea aperiódica o esporádica, exactamente τ_0 unidades de tiempo antes de terminar su período; el mismo instante de tiempo en que las tareas periódicas J_i, $1 \leq i \leq n$, sin incluir a J_0, se encuentran en un instante crítico. Por si fuera poco, la actividad aperiódica requiere una nueva

activación del SPAA al inicio de su siguiente período, por lo que se hacen dos activaciones consecutivas de J_0 retrasando aún más la ejecución del resto de tareas periódicas.

Si tomamos como punto de referencia el instante de tiempo en que ocurren los eventos (a) y (b), y lo consideramos $t = 0$, entonces $I_o = \tau_0$ e $I_i = 0$, $1 \leq i \leq n$. La ejecución de J_0 se dará en los intervalos $[0, \tau_0]$, $[\tau_0, 2\tau_0]$, $[\tau_0 + \pi_0, 2\tau_0 + \pi_0]$, etc. Si la primera ejecución de cada tarea periódica cumple su plazo de respuesta en estas condiciones, también cumplirán su plazo las activaciones subsecuentes. La primera activación de la tarea J_k cumplirá su plazo de respuesta (π_k), si y sólo si existe un tiempo t, menor a π_k, en el que todo el trabajo de prioridad mayor a k esté terminado y J_k se haya ejecutado τ_k unidades de tiempo. La demanda total de tiempo de procesamiento cuando las tareas tienen su tiempo de respuesta más largo, para $t \geq \pi_0$, está dada por:

$$\tau_0 + \tau_0 \left\lceil \frac{t - \pi_0}{\pi_0} \right\rceil + \sum_{j=1}^{k} \tau_j \left\lceil \frac{t}{\pi_j} \right\rceil, \tag{4.2}$$

donde los primeros dos términos corresponden al máximo tiempo de ejecución posible de J_0, y el último se refiere a la demanda acumulada de tiempo de procesamiento de las tareas J_i, $1 \leq i \leq k$. Entonces J_k cumplirá su plazo de respuesta si:

$$\min_{\pi_0 \leq t \leq \pi_k} \left\{ \tau_0 \cdot \left(1 + \left\lceil \frac{t - \pi_0}{\pi_0} \right\rceil \right) + \sum_{j=1}^{k} \tau_j \left\lceil \frac{t}{\pi_j} \right\rceil \right\} \leq t, \tag{4.3}$$

o de manera equivalente,

$$\min_{\pi_0 \leq t \leq \pi_k} \left\{ \frac{1}{t} \left(\tau_0 \cdot \left(1 + \left\lceil \frac{t - \pi_0}{\pi_0} \right\rceil \right) + \sum_{j=1}^{k} \tau_j \left\lceil \frac{t}{\pi_j} \right\rceil \right) \right\} \leq 1, \tag{4.4}$$

La minimización de la ecuación (4.4) se hace sobre la variable continua $t \in [\pi_0, \pi_k]$. Sin embargo, t tiene que verificarse exclusivamente un número finito de veces: sólo en aquellos puntos en que la expresión es discontinua, lo cual ocurre en los tiempos t que son múltiplos de π_i, para toda $1 \leq i \leq k$ [Lehoczky et al., 89] [Strosnider et al., 95]. Modificando (4.4) se obtiene:

$$\min_{(t\in H_k)}\left\{\frac{1}{t}\left(\tau_0\cdot\left(1+\left\lceil\frac{t-\pi_0}{\pi_0}\right\rceil\right)+\sum_{j=1}^{k}\tau_j\left\lceil\frac{t}{\pi_j}\right\rceil\right)\right\}\le 1, \tag{4.5}$$

con $H_k = \{i\cdot\pi_j \mid j = 1, \ldots, k\ ;\ i = 1, \ldots, \lfloor\pi_k/\pi_j\rfloor\}$.

Para que el conjunto de tareas sea planificable, la desigualdad (4.5) debe cumplirse para k, $1 \le k \le n$. Entonces, la condición necesaria y suficiente para planificar J y un SPAA es:

$$\max_{1\le k\le n}\min_{t\in H_k}\left\{\frac{1}{t}\left(\tau_0\cdot\left(1+\left\lceil\frac{t-\pi_0}{\pi_0}\right\rceil\right)+\sum_{j=1}^{k}\tau_j\left\lceil\frac{t}{\pi_j}\right\rceil\right)\right\}\le 1. \tag{4.6}$$

En nuestro caso, el conjunto de tareas periódicas puede tener restricciones de recursos, al menos en su parte obligatoria, lo cual puede producir que algunas tareas periódicas sean bloqueadas durante algún tiempo por otras tareas de menor prioridad. Considerar el protocolo de techo de prioridad permite que los efectos de un bloqueo sean predecibles. Sin embargo, (4.6) aún no considera el efecto del bloqueo en la planificabilidad.

[Sha et al., 90] hicieron un análisis de planificabilidad y establecieron un conjunto de condiciones suficientes en las que un conjunto de tareas periódicas que usan el protocolo de techo de prioridad pueden ser planificadas por el algoritmo de razón monotónica. Este trabajo nos permitirá extender la ecuación (4.5).

Sea β_i el conjunto de secciones críticas accedidas por cualquier tarea de menor prioridad que J_i, que correspondan a un semáforo binario cuyo techo de prioridad es mayor o igual a la prioridad de J_i. Este es el conjunto de secciones críticas que pueden bloquear una activación de J_i. Considerando que las secciones críticas están anidadas adecuadamente, el conjunto está parcialmente ordenado por inclusión de conjuntos. Entonces, aprovechando esta característica, pueden eliminarse las secciones críticas que sean parte de una sección crítica mayor. β_i puede reducirse a β_i^*, el conjunto de las secciones críticas más largas de las tareas de menor prioridad que pueden bloquear una activación de J_i. De acuerdo a [Sha et al., 90], una activación de J_i puede ser bloqueada cuando mucho por un solo elemento de β_i^*. Por tanto, en el peor de los casos, una activación de J_i puede ser bloqueada, cuando

mucho, durante la duración del elemento más largo de β_i^*. A este peor de lo casos del tiempo de bloqueo le llamaremos B_i.

Para probar si una tarea J_k, $1 \leq k \leq n$, que tiene restricciones de recursos, puede ser planificada es necesario considerar el tiempo que otras tareas de menor prioridad podrían mantenerla bloqueada. Modificando la ecuación (4.5) para incluir el efecto del bloqueo se obtiene que la tarea J_k puede ser planificada si:

$$\min_{t \in H_k} \left\{ \frac{1}{t} \left(\tau_0 \cdot \left(1 + \left\lceil \frac{t - \pi_0}{\pi_0} \right\rceil \right) + \sum_{j=1}^{k-1} \tau_j \left\lceil \frac{t}{\pi_j} \right\rceil + \tau_k + B_k \right) \right\} \leq 1. \tag{4.7}$$

donde B_k representa las unidades de tiempo que en el peor de los casos se retrasará por bloqueo la ejecución de τ_k. En la ecuación (4.7) es posible extraer el último término de la sumatoria original de la ecuación (4.5) porque siendo π_k el período de mayor longitud, y sin importar cual sea el valor que tome $t \in H_k$, cuando $j = k$, $\tau_j \lceil t / \pi_j \rceil$ siempre será igual a τ_k, pues la función techo siempre producirá un valor de 1. No es necesario considerar tiempo de bloqueo para el SPAA, pues no comparte recursos con ninguna otra tarea, y por tanto siempre podrá remover de ejecución a cualquier tarea y nunca estará bloqueado.

Evaluar si con el algoritmo de razón monotónica puede planificarse un servidor periódico de actividad aperiódica J_o, junto con un conjunto de tareas periódicas J que tienen restricciones de recursos y utilizan el protocolo de techo de prioridad, requiere evaluar la ecuación (4.7) para toda k, $1 \leq k \leq n$. Dicha planificación puede hacerse si se cumple que:

$$\max_{1 \leq k \leq n} \min_{t \in H_k} \left\{ \frac{1}{t} \left(\tau_0 \cdot \left(1 + \left\lceil \frac{t - \pi_0}{\pi_0} \right\rceil \right) + \sum_{j=1}^{k-1} \tau_j \left\lceil \frac{t}{\pi_j} \right\rceil + \tau_k + B_k \right) \right\} \leq 1, \tag{4.8}$$

con $H_k = \{ i \cdot \pi_j \mid j = 1, \ldots, k \; ; \; i = 1, \ldots, \lfloor \pi_k / \pi_j \rfloor \}$. Así pues, la ecuación (4.8) representa la prueba de planificabilidad que debe cumplir un conjunto de tareas periódicas que comparten recursos y un SPAA, que actúa sobre demanda, para garantizar que no se perderá ningún plazo de respuesta.

La ecuación (4.9) es útil si se quiere evaluar la planificabilidad del conjunto de tareas sin considerar un SPAA.

$$\max_{1\le k\le n} \min_{t\in H_k} \left\{ \frac{1}{t} \left(\sum_{j=1}^{k-1} \tau_j \left\lceil \frac{t}{\pi_j} \right\rceil + \tau_k + B_k \right) \right\} \le 1, \tag{4.9}$$

4.6.2 Diseño del servidor periódico de actividad aperiódica

Hasta ahora se ha hecho mención del período y capacidad de procesamiento del SPAA sin especificar la manera en que se determina el valor de estos parámetros. En esta sección se discute este tópico.

Como se mencionó con anterioridad, una de las características del SPAA es que debe ser la tarea con período más corto y, en consecuencia, la tarea de prioridad más alta. Esta característica le permite al servidor atender de inmediato la llegada de las tareas aperiódicas o esporádicas sin estar sujeto a interrupciones por parte de las tareas periódicas.

Mientras el período π_0 del SPAA sea menor al período π_1 de la tarea periódica de menor prioridad J_1, varios son los posibles valores que puede tomar π_0. Sin embargo, como lo explican [Strosnider et al., 95], es preferible que el valor seleccionado sea el más grande posible, pues esto resulta en un menor desperdicio de la capacidad del servidor periódico de actividad aperiódica. Por ejemplo, supónganse dos servidores periódicos $J_{0,1}$ y $J_{0,2}$, tales que $\tau_{0,1} = 1$, $\pi_{0,1} = 10$ y $\tau_{0,2} = 2$, $\pi_{0,2} = 20$. $J_{0,2}$ puede proporcionar una capacidad de dos unidades de procesamiento en cualquier momento entre los tiempos *0* y *20*. En el tiempo *20* se pierde toda la capacidad no utilizada. $J_{0,1}$ también proporciona una capacidad de procesamiento de dos unidades en el intervalo *[0, 20]*, sin embargo, una de ellas se perderá en el tiempo *10* si no es utilizada en el intervalo *[0, 10]*.

Encontrar la capacidad máxima de procesamiento que puede tomar el servidor es una actividad más complicada que determinar el período y requiere el uso de la prueba de planificabilidad desarrollada en la sección previa. La complejidad asociada con analizar en línea la prueba de planificabilidad de la ecuación (4.8) podría ser prohibitiva para algunas aplicaciones de tiempo real, por lo que el diseño del SPAA deberá hacerse fuera de línea.

Para cada selección del período π_0 de J_0, tal que $\pi_0 < \pi_1$, existe una capacidad de procesamiento τ_0 máxima dada por la prueba de planificabilidad de la ecuación (4.8). En general, conforme π_0 crece, también lo hace el valor de τ_0. Si se selecciona como se sugiere, el mayor valor posible para π_0, se puede obtener el valor máximo de τ_0 que mantiene planificable el conjunto de tareas periódicas y el servidor de actividad aperiódica.

Una vez encontrado el valor de τ_0, se requiere compararlo contra un estimado de la capacidad de procesamiento necesaria para atender las tareas aperiódicas y esporádicas de una manera eficiente. Si el valor de τ_0 es satisfactorio, las tareas periódicas serán ejecutadas completas sin que pierdan su plazo y se espera que también las aperiódicas y esporádicas se ejecuten completas. Sin embargo, es probable que τ_0 no sea satisfactorio, en cuyo caso habrá que calcular un nueva capacidad de procesamiento para el servidor, considerando solamente las partes obligatorias tanto de las tareas periódicas como de las aperiódicas y esporádicas. A continuación, a través de un ejemplo, se describe la manera en que se realiza el diseño de un SPAA.

4.6.2.1 Ejemplo del diseño de un SPAA

Supongamos que las tareas periódicas del conjunto $J = \{J_1, J_2, J_3\}$ están construidas de la siguiente manera:

J_1: **NOP, P(S_1), NOP, V(S_1), NOP,** *NOP*

J_2: **P(S_1), NOP, V(S_1), P(S_2), V(S_2), NOP,** *NOP, NOP*

J_3: **P(S_3), NOP, P(S_2), NOP, V(S_2), V(S_3), NOP,** *NOP, NOP, NOP*

Las instrucciones P y V son las primitivas para el control de concurrencia a través del manejo de semáforos binarios. Las instrucción NOP representa la ejecución de cualquier operación o instrucción válida. S_i es el semáforo asociado al recurso R_i. Cada tarea está dividida en 2 partes: una obligatoria (señalada en negritas) y una opcional (indicada en itálicas). Supongamos también que la ejecución de cada instrucción toma una unidad de tiempo. Las características del conjunto de tareas periódicas se detallan en la Tabla 4.7.

Tabla 4.7 Características del conjunto de tareas periódicas del ejemplo de la sección 4.6.2.1.

Tarea	*Período (π_i)*	*Tiempo de ejecución (τ_i)*	*Parte obligatoria (m_i)*	*Parte opcional (o_i)*
J_1	30	6	5	1
J_2	35	8	6	2
J_3	40	10	7	3

Determinación del período del SPAA

Puesto que el período más corto de una tarea es *30*, el período máximo posible que garantiza que el SPAA tenga la prioridad más alta, con base en el algoritmo de razón monotónica, es $\pi_0 = 29$. Este será el período seleccionado.

Cálculo de la capacidad de procesamiento del SPAA

Para encontrar la capacidad de procesamiento del SPAA se tratará de planificar cada uno de los siguientes casos hasta encontrar uno que sea viable:

1. Tareas periódicas completas - tareas aperiódicas y esporádicas completas.
2. Tareas periódicas completas - subtareas aperiódicas y esporádicas obligatorias.
3. Subtareas periódicas obligatorias - tareas aperiódicas y esporádicas completas.
4. Subtareas periódicas obligatorias - subtareas aperiódicas y esporádicas obligatorias.

El cálculo de la capacidad de procesamiento τ_0 del SPAA (J_0) se hace a partir de la ecuación (4.8), por lo que es necesario calcular los tiempos de bloqueo B_i que pueden sufrir las partes obligatorias de las tareas periódicas en el peor de los casos y el conjunto instantes de decisión H_i para cada tarea J_i.

La asignación de prioridades que usa LAT está basada en el algoritmo de razón monotónica. De acuerdo con este enfoque, J_0 tiene prioridad *0*, y J_1, J_2 y J_3 tienen prioridades *1*, *2* y *3*, respectivamente. Con base en esta asignación de prioridades, los techos de prioridad de los semáforos S_1, S_2 y S_3 son *1*, *2* y *3*, respectivamente.

Sea $z_{i,j}$ la sección crítica de la tarea i en el recurso j. A partir de la definición de las tareas se obtiene el conjunto de secciones críticas más largas que pueden bloquear una activación de J_i: $\beta_1^* = \{z_{2,1}\}$, $\beta_2^* = \{z_{3,2}\}$ y $\beta_3^* = \{\}$. De la duración de las secciones críticas en estos conjuntos se obtienen los siguientes tiempos máximos de bloqueo para cada tarea periódica: $B_1 = 3$, $B_2 = 3$ y $B_3 = 0$.

Los conjuntos de instantes de decisión para las tareas periódicas son: $H_1 = \{30\}$, $H_2 = \{30, 35\}$ y $H_3 = \{30, 35, 40\}$.

Caso 1. Tareas periódicas completas - tareas aperiódicas y esporádicas completas

Encontrar el valor máximo de τ_0 que con período $\pi_0 = 29$ mantiene planificable el conjunto de tareas periódicas completas y el servidor periódico requiere evaluar varios valores candidatos. En particular, el valor de τ_0 estará en el intervalo $[0, \pi_0]$. Una estrategia es buscar el valor de τ_0 de una manera similar a una búsqueda binaria, pero para ello deben considerarse tiempos discretos dentro del intervalo continuo $[0, \pi_0]$. En este caso particular en que la ejecución de una instrucción toma una unidad de tiempo, la capacidad de τ_0 debe ser un múltiplo de una unidad, por lo que su valor debe encontrarse en los elementos de la secuencia $\boldsymbol{K} = 0, 1, 2, 3, \ldots, 29$. En otros casos deben considerarse los tiempos de procesamiento del conjunto de instrucciones para identificar los tiempos discretos que son candidatos. Por ejemplo, cuando la ejecución de una instrucción tome 0.2 unidades de tiempo, el valor máximo de τ_0 debe buscarse en los elementos de la secuencia $\boldsymbol{K} = 0, 0.2, 0.4, 0.8, 1, 1.2, \ldots \pi_0$.

Sea K_h el elemento en la *h-ésima* posición de $\boldsymbol{K}$, para $0 \leq h \leq m-1$, donde m es la longitud de la secuencia. La máxima capacidad posible de τ_0 puede obtenerse con el siguiente algoritmo, donde el resultado queda depositado en la variable τ_{act}:

Algoritmo 4.3. Algoritmo de búsqueda de τ_0.

límite_inferior = 0
límite_superior = m-1
τ_{act} = 0
mientras(límite_superior > límite_inferior) **hacer**
    medio = **entero**((límite_inferior + límite_superior) / 2)
    Si se cumple la ecuación (4.8) para $\tau_0 = K_{medio}$
        $\tau_{act} = K_{medio}$
        límite_inferior = medio + 1
    si no
        límite_superior = medio - 1
fin mientras

Para el ejemplo, el algoritmo se comporta de la siguiente manera. Se hacen *límite_inferior = 0*, *límite superior = 29* y $\tau_{act} = 0$. La condición del ciclo se cumple, por lo que se calcula el valor de medio: *medio = (límite_inferior + límite_superior) / 2 = (0 + 29) / 2 = 14*. Evaluando la ecuación (4.8) para $\tau_0 = K_{14} = 14$, se tiene:

Cuando $k = 1$, t sólo puede tomar el valor de *30*, por lo que

$$\min_{t\in H_k}\left\{\frac{1}{t}\left(\tau_0\cdot\left(1+\left\lceil\frac{t-\pi_0}{\pi_0}\right\rceil\right)+\sum_{j=1}^{k-1}\tau_j\left\lceil\frac{t}{\pi_j}\right\rceil+\tau_k+B_k\right)\right\}=\left\{\frac{1}{30}\left(14\cdot\left(1+\left\lceil\frac{30-29}{29}\right\rceil\right)+6+3\right)\right\}=1.2333$$

Cuando $k = 2$, t toma los valores *30* y *35*. Para $k = 2$ y $t = 30$ se tiene

$$\left\{\frac{1}{t}\left(\tau_0\cdot\left(1+\left\lceil\frac{t-\pi_0}{\pi_0}\right\rceil\right)+\sum_{j=1}^{k-1}\tau_j\left\lceil\frac{t}{\pi_j}\right\rceil+\tau_k+B_k\right)\right\}=\left\{\frac{1}{30}\left(14\cdot\left(1+\left\lceil\frac{30-29}{29}\right\rceil\right)+6\left\lceil\frac{30}{30}\right\rceil+8+3\right)\right\}=1.5$$

Para $k = 2$ y $t = 35$

$$\left\{\frac{1}{t}\left(\tau_0\cdot\left(1+\left\lceil\frac{t-\pi_0}{\pi_0}\right\rceil\right)+\sum_{j=1}^{k-1}\tau_j\left\lceil\frac{t}{\pi_j}\right\rceil+\tau_k+B_k\right)\right\}=\left\{\frac{1}{35}\left(14\cdot\left(1+\left\lceil\frac{35-29}{29}\right\rceil\right)+6\left\lceil\frac{35}{30}\right\rceil+8+3\right)\right\}=1.4571$$

El mínimo de los dos valores calculados es *1.4571*. Se sigue el mismo procedimiento para $k = 3$. Sin embargo, desde el paso en que $k = 1$ podemos saber que cuando $\tau_0 = 14$ el conjunto J y el SPAA no son planificables, pues el valor calculado en el término izquierdo de la ecuación (4.8) al menos será igual a *1.2333*, el cual no cumple la condición de ser menor que *1*. Obviamente la prueba puede terminarse en ese paso para hacer más eficiente la búsqueda del valor máximo posible de τ_0, sin embargo, aquí se prosiguió para mostrar la manera en que se evalúa la ecuación (4.8).

Puesto que no se pasó la prueba de planificabilidad, el algoritmo hace *límite_superior = medio - 1 = 13*. La siguiente iteración del ciclo hará la prueba de planificación para $\tau_0 = 6$. Luego de varias iteraciones, el algoritmo obtiene que el valor máximo posible para τ_0 es *2*.

La capacidad de procesamiento calculada para el SPAA debe ser comparada contra un estimado Q de la *capacidad de procesamiento necesaria del SPAA* para atender la actividad aperiódica y esporádica. Q es un valor que sirve como referencia para determinar que tan satisfactorio es el valor τ_0 encontrado. Q es sólo una referencia porque el comportamiento exacto de las tareas aperiódicas no es conocido, aunque se asume que se ajusta a ciertas distribuciones probabilísticas.

Supóngase que en el sistema se presentan dos tareas esporádicas con las características definidas en la Tabla 4.8.

Tabla 4.8 Tareas esporádicas del ejemplo de la sección 4.6.2.1

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	3	2	1	25
E_2	5	3	2	70

Supóngase también que en el sistema se presentan tareas aperiódicas con plazos de respuesta arbitrarios cuyas llegadas forman un proceso Poisson con razón media de llegadas λ igual a 0.02. Si el proceso de llegadas es Poisson, existe una variable asociada llamada tiempo medio entre arribos $(1/\lambda)$ que sigue una distribución exponencial [Gross y Harris, 85]. En este caso, $1/\lambda = 50$, es decir, se espera la llegada de una tarea aperiódica cada *50* unidades de tiempo. Los tiempos de ejecución de las subtareas obligatorias y opcionales de las tareas aperiódicas siguen una distribución exponencial con medias $1/\mu_m = 2$ y $1/\mu_o = 1$, respectivamente.

El estimado de capacidad de procesamiento necesaria del SPAA para atender la actividad aperiódica y esporádica Q es la suma de la capacidad necesaria estimada para la atención de las tareas esporádicas E y la capacidad necesaria estimada para la atención de las tareas aperiódicas A: $Q = A + E$.

Si los plazos de respuesta de las tareas esporádicas están determinados por su mínimo tiempo entre arribos, es decir, el plazo de una tarea esporádica es igual a la suma de su tiempo de llegada y su mínimo tiempo entre arribos, se puede calcular la capacidad necesaria E, en un período del SPAA para atender todas las tareas esporádicas que lleguen, sin perder algún plazo. En el peor de los casos, las tareas esporádicas llegarán siempre en su mínimo tiempo entre arribos. Suponiendo que eso ocurre, dicha capacidad puede ser calculada de la siguiente manera:

1. Para π_0 conocido, calcular el número de veces f_i que cabe el período del SPAA en el plazo de respuesta de la tarea esporádica E_i: $f_i = mta_i / \pi_0$.

   Para el ejemplo en cuestión $f_1 = mta_1 / \pi_0 = 25 / 29 = 0.8621$ y $f_2 = mta_2 / \pi_0 = 70 / 29 = 2.4138$. f_i representa el número de períodos del SPAA transcurridos antes de llegar al plazo de las tareas esporádicas.

2. Dividir el tiempo de ejecución C_i de la tarea esporádica E_i entre el valor f_i calculado para determinar la capacidad de procesamiento por período q_i requerida por cada esporádica. En otras palabras, se prorratea el tiempo de ejecución de cada tarea esporádica entre el número de veces que se ejecutará el SPAA antes de llegar al plazo de la tarea. Para el ejemplo: $q_1 = C_1 / f_1 = 3 / 0.8621 = 3.4799$ y $q_2 = C_2 / f_2 = 5 / 2.4138 = 2.0714$.

3. La sumatoria de todos los valores de q_i es el valor E buscado. $E = q_1 + q_2 = 3.4799 + 2.0714 = 5.5513$. Esto es, para la atención de las tareas esporádicas del ejemplo en cuestión, el SPAA debe tener una capacidad de procesamiento mínima de *5.5513* unidades de tiempo en cada período.

El cálculo de la capacidad de procesamiento necesaria para la atención de las tareas aperiódicas A requiere un análisis basado en resultados conocidos de la teoría de colas. En este análisis debe calcularse la longitud promedio del período ocupado de las tareas aperiódicas *(aperiodic busy period)* sin considerar la presencia de las tareas periódicas y esporádicas. Un período ocupado inicia con la llegada de un cliente a un canal ocioso y termina cuando el canal vuelve a estar ocioso [Gross y Harris, 85]. Calcular el período ocupado de las tareas aperiódicas permite estimar el tiempo necesario adicional a E con que debe contar el SPAA en cada período.

Si los arribos de las tareas aperiódicas forman un proceso Poisson y tienen un requerimiento de servicio promedio $E(S)$, entonces, asumiendo que las tareas aperiódicas no son interrumpidas por periódicas y esporádicas, la longitud promedio del período ocupado está dada por $E(B) = E(S)/(1-\rho)$, donde ρ es la intensidad de tráfico de las tareas aperiódicas [Strosnider et al., 95]. La intensidad de tráfico es el número esperado de arribos por tiempo de servicio promedio $\rho = (\lambda/\mu)$ y representa la utilización promedio del procesador [Klein et al., 93][Gross y Harris, 85].

$E(S)$ es el tiempo de ejecución promedio requerido por las tareas aperiódicas, es decir, $E(S) = 1/\mu$. Entonces,

$$E(B) = \frac{E(S)}{1-\rho} = \frac{\frac{1}{\mu}}{1-\frac{\lambda}{\mu}} = \frac{1}{\mu - \lambda} \tag{4.10}$$

Una vez que se ha calculado la longitud del período ocupado, es importante representar en el cálculo de A la relación existente entre el π_o seleccionado y el tiempo medio entre arribos $(1/\lambda)$ de las tareas aperiódicas. Pueden presentarse dos casos :

- $\pi_o > 1/\lambda$. Esto significa que durante un período del SPAA pueden presentarse varios períodos ocupados de tareas aperiódicas. Se esperan aproximadamente $\pi_o / (1/\lambda) = \lambda\pi_o$ períodos ocupados. Si este es el caso, el valor buscado de A es $A = \lambda\pi_o E(B)$.
- $\pi_o \leq 1/\lambda$. Esto quiere decir que cuando mucho se espera un período ocupado por cada período del SPAA. En este caso, $A = E(B)$.

Para el ejemplo en cuestión se definieron las medias de los tiempos de ejecución de las partes obligatorias $(1/\mu_m)$ y opcionales $(1/\mu_o)$ de las tareas aperiódicas. Para cuestiones del cálculo de A, se supondrá que el tiempo de ejecución de la tarea completa sigue una distribución exponencial con media $1/\mu = 1/\mu_m + 1/\mu_o$. Entonces $1/\mu = 2 + 1 = 3$; $\mu = 1/3$. Estrictamente, el caso más simple de la suma de dos variables independientes que siguen una distribución exponencial es cuando ambas tienen el mismo parámetro λ, lo cual produce una distribución gama con parámetros λ y 2 [Meyer, 86]. Sin embargo, si los parámetros de las dos distribuciones son diferentes, lo cual es lo más probable en los casos prácticos, no se puede garantizar el resultado descrito. Por otro lado, para aplicar los resultados conocidos de la teoría de colas, sería deseable que la distribución resultante fuese una distribución exponencial, lo cual no puede asegurarse. Tomando en cuenta que Q es sólo un estimado que permite darse una idea de la capacidad de procesamiento necesaria del SPAA, se aceptará la suposición hecha, verificando su comportamiento en el capítulo de resultados.

De acuerdo con los parámetros π_o y λ, $A = E(B) = 1 / (0.3333 - 0.02) = 3.1915$. Una vez conocidos los valores estimados para E y A, se obtiene $Q = E + A = 5.5513 + 3.1915 = 8.7428$.

En general, se espera que si $\tau_o > Q$ las tareas aperiódicas y esporádicas se ejecuten con niveles muy bajos de error y que las tareas rechazadas por no cumplir la MFC sean muy pocas. Además, las tareas periódicas, incluyendo sus partes obligatoria y opcional, son planificables con un error promedio de 0. Sin embargo, en el ejemplo que se ha ido desarrollando, $\tau_o < Q$ $(2 < 8.7428)$. Esto quiere decir que τ_o no es suficiente para atender las tareas aperiódicas y esporádicas completas.

Caso 2. Tareas periódicas completas - subtareas aperiódicas y esporádicas obligatorias

Se puede realizar el cálculo de Q tomando en cuenta solamente los tiempos de ejecución de las partes obligatorias de las tareas aperiódicas y esporádicas en lugar de su tiempo de ejecución total. El valor Q' estimado de esta forma será utilizado como una referencia de la *capacidad de procesamiento mínima aceptable del SPAA*. Evidentemente, este valor producirá un mayor número de tareas rechazadas por no cumplir la MFC, pero seguirá produciendo un plan que minimice el error total de las tareas aperiódicas y esporádicas.

Q' es la suma de E' y A'. $E' = q_1' + q_2' = ob_1/f_1 + ob_2/f_2 = 2 / 0.8621 + 3 / 2.4138 = 3.5628$. $A' = 1 / (\mu_m - \lambda) = 1 / (0.5 - 0.02) = 2.0833$. $Q' = E' + A' = 3.5628 + 2.0833 = 5.6461$.

Aún así, $\tau_o < Q'$ $(2 < 5.6461)$. La capacidad de procesamiento del SPAA no es suficiente.

Caso 3. Subtareas periódicas obligatorias - tareas aperiódicas y esporádicas completas

La alternativa es probar la planificabilidad de las partes obligatorias de las tareas periódicas para encontrar un valor mayor de τ_o que permita atender las tareas aperiódicas y esporádicas completas. Esto implica que sólo las partes obligatorias de las tareas periódicas aseguran su planificabilidad; las partes opcionales se ejecutan lo mejor posible de acuerdo a la política del algoritmo LAT. Aplicando el algoritmo de búsqueda de τ_o y sustituyendo el tiempo de ejecución de cada tarea τ_i por el tiempo de ejecución de su subtarea obligatoria m_i en la ecuación (4.8) se obtiene una *capacidad de procesamiento máxima posible del SPAA* $\tau_o' = 6$. Aún así, el valor de τ_o' es menor al estimado Q $(6 < 8.7428)$.

Caso 4. Subtareas periódicas obligatorias - Subtareas aperiódicas y esporádicas obligatorias

Finalmente, se consideran solamente las partes obligatorias de las tareas periódicas, aperiódicas y esporádicas. Con base en los cálculos previos $\tau_o' > Q'$ $(6 > 5.6461)$; la capacidad de procesamiento máxima posible del SPAA se ajusta con el estimado de la capacidad de procesamiento mínima aceptable. Por tanto, $\tau_o' = 6$ y $\pi_o = 29$ caracterizan al SPAA.

4.6.2.2 Comportamiento esperado del método PTF

En el ejemplo desarrollado se presentaron algunas situaciones en que la capacidad del servidor periódico no era satisfactoria para atender la actividad de las tareas aperiódicas y esporádicas. En general, se espera que se presenten los siguientes casos en el comportamiento del método PTF:

a) $\tau_o > Q$. Las tareas periódicas completas son planificables sin error. Se espera que las tareas aperiódicas y esporádicas se ejecuten con niveles muy bajos de error y que las tareas rechazadas por no cumplir la restricción de viabilidad de la parte obligatoria sean muy pocas.

b) $Q' \leq \tau_o \leq Q$. Las tareas periódicas completas son planificables sin error. Se incrementan el número de tareas aperiódicas y esporádicas que son rechazadas por no cumplir la MFC, y su error total producido. Sin embargo, la capacidad del servidor periódico de actividad aperiódica es aceptable. Aunque las tareas periódicas, aperiódicas y esporádicas son todas importantes, podría decirse que se está dando preferencia a la atención de la actividad periódica.

c) $\tau_o < Q'$. La capacidad del servidor periódico no es aceptable. Será necesario considerar solamente las partes obligatorias de las tareas periódicas para obtener una mayor capacidad de procesamiento del SPAA y mejorar la atención de las tareas aperiódicas y esporádicas.

d) Es probable que se presente el caso (b) y que los valores de τ_o y Q' sean iguales o muy próximos. También es probable que cuando $\tau_o \leq Q$ no se quiera dar preferencia a la atención de las tareas periódicas, sino buscar un compromiso entre la atención de las periódicas, aperiódicas y esporádicas. En estas situaciones sería conveniente aplicar la misma estrategia que en el caso (c).

Si se ha de considerar sólo la parte obligatoria de las tareas periódicas para determinar la capacidad de procesamiento del SPAA, como se sugiere en los casos (c) y (d) podrían presentarse los siguientes casos:

e) $\tau_o' > Q$. Si uno desea dar preferencia a las tareas aperiódicas y esporádicas esto es perfecto, aunque las tareas periódicas verán aumentado su error promedio considerablemente. Si no, podría considerarse la estrategia utilizada en el caso (f).

f) $Q' \leq \tau_o' \leq Q$. Si el valor de τ_o' es igual o muy cercano a Q las tareas aperiódicas y esporádicas tendrán una buena atención, pero las periódicas podrían estar teniendo un error

promedio considerable. Si este es el caso, valdría la pena reducir un poco el valor de τ_o' para establecer un compromiso entre la atención a las tareas periódicas, aperiódicas y esporádicas. Una herramienta que simule el comportamiento del sistema será muy útil en estos casos, pues permitirá observar como se comportan las tareas cuando el SPAA tiene diferentes capacidades de procesamiento y así decidir cual de ellas parece más conveniente.

g) $\tau_o' < Q'$. Se tiene una capacidad de atención de tareas periódicas y esporádicas muy pobre. Ni siquiera la cualidad de las tareas de ser flexibles asegura un comportamiento aceptable del sistema. Será necesario buscar equipos con mayor capacidad de procesamiento.

4.7 Complejidad del método de planificación

Puesto que en un sistema de tiempo real el recurso más importante es el tiempo, es de interés calcular la complejidad temporal del método de planificación desarrollado. Específicamente, interesa conocer el costo de administración producido. Por simplicidad se supondrá que el tiempo utilizado en los cambios de contexto es tan pequeño que es despreciable.

La parte del método dedicada al manejo de la actividad aperiódica es la más compleja, por ello este análisis se centra en ese tópico. El manejo de la actividad periódica es simple, y el costo de administración más importante es el que se produce en las inserciones ordenadas de las tareas en la lista de tareas listas para ejecución. En el peor de los casos estas inserciones requieren una búsqueda secuencial de la posición de las tareas, lo cual tiene una complejidad de $O(n)$, donde n es el número de tareas que se encuentran actualmente en la lista de tareas listas para ejecución.

Es en la llegada de cada tarea aperiódica o esporádica donde se realizan las operaciones de actualización de la lista de reservaciones e inserción de tareas en la cola de actividad aperiódica. Supongamos que la lista de reservaciones está definida como una lista doblemente ligada de nodos o un arreglo de estructuras, cada uno de ellos representando la capacidad de procesamiento del SPAA en un período, y un indicador de la cantidad de tiempo reservado para cada una de las subtareas obligatorias aún no terminadas. Los intervalos reservados se especifican manteniendo sus instantes de inicio y término, dentro de los nodos de la lista. La Figura 4.7 presenta, de manera gráfica, una lista de reservaciones implantada en una lista doblemente ligada, para un SPAA con período $\pi_0 = 20$

y capacidad de procesamiento $\tau_0 = 5$, cuyos siguientes inicios de período se dan en los instantes *50* y *70*. En el ejemplo, se supone la existencia de dos tareas cuyas subtareas obligatorias M_1 y M_2 aún no terminadas son *3* y *4* unidades de tiempo, respectivamente.

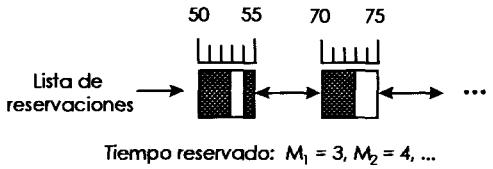


Figura 4.7 Ejemplo de lista de reservaciones.

El costo de administración t_{ov} producido por la llegada de una nueva tarea está determinado por la siguiente fórmula:

$$t_{ov} = t_{alr} + t_{un} + t_{MFC} + t_r + t_{itc} \tag{4.11}$$

donde:

t_{alr} es el tiempo requerido para que la tarea aperiódica o esporádica en el inicio de la cola de actividad aperiódica actualice su reservación en la llegada de una nueva tarea aperiódica o esporádica.

t_{un} corresponde al tiempo necesario para detectar el nodo en que debe iniciar la reservación inversa para la nueva tarea.

t_{MFC} es el tiempo necesario para verificar si la parte obligatoria de la tarea recién llegada es planificable.

t_r es el tiempo que toma efectuar la reservación para la nueva tarea en la lista de reservaciones.

t_{itc} es el tiempo que tarda la inserción de la nueva tarea en la cola de actividad aperiódica.

Para determinar la complejidad del costo de administración conviene analizar el peor de los casos en la llegada de una tarea aperiódica o esporádica. Sea n el número de tareas presentes en la cola de actividad aperiódica, es decir, el número de tareas aperiódicas y esporádicas previamente aceptadas que aún no han sido terminadas. El peor de los casos ocurre cuando, una vez hecha la actualización de la reservación de la tarea del inicio de la cola de actividad aperiódica, las n tareas producen una lista de reservaciones como la que se muestra en la Figura 4.8 y la subtarea obligatoria de la tarea recién llegada T_{n+1} requiere todo el tiempo intermedio entre los intervalos reservados y algún tiempo extra hacia ambos extremos. En este caso, cada una de las tareas aperiódicas tiene una reservación separada de las demás, por lo que en la lista de reservaciones se presentan n intervalos reservados.

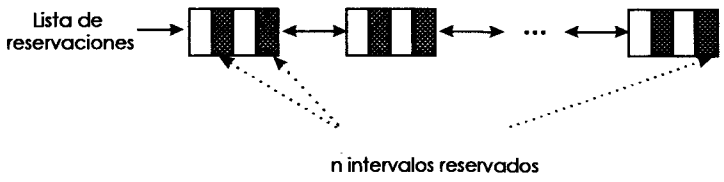


Figura 4.8. Lista de reservaciones en el peor de los casos.

Por simplicidad, en el análisis que se describe a continuación los tiempos de inserción, borrado y recorrido de los nodos de la lista de reservaciones se consideran despreciables.

El tiempo t_{alr} requerido para que la tarea aperiódica T_i en el inicio de la cola de actividad aperiódica actualice su reservación no es función del número de tareas en la cola, sino del tamaño de la parte obligatoria de la tarea. La actualización se hace agregando o borrando un intervalo reservado en el inicio de la lista de reservaciones. Esta actualización puede requerir modificar uno o varios nodos de la lista de reservaciones en caso de que el intervalo reservado utilice varios nodos. En el peor de los casos se requerirá modificar $\lceil m_i / \tau_0 \rceil + 1$ nodos, realizando las operaciones para actualizar el inicio y término de cada uno de los intervalos reservados. Por ejemplo, si $m_i = 10$ y $\tau_0 = 5$, cuando mucho se deberán modificar *3* nodos, como se aprecia en la Figura 4.9, en donde se reservan *10* unidades de tiempo en la lista de reservaciones. Si los tiempos de procesamiento de la parte obligatoria de las tareas aperiódicas y esporádicas son conocidos o siguen una distribución conocida, puede obtenerse un valor estimado del tiempo que tomará la actualización, a partir del valor conocido o la media de la distribución. Este tiempo puede considerarse constante o de *O(1)*.

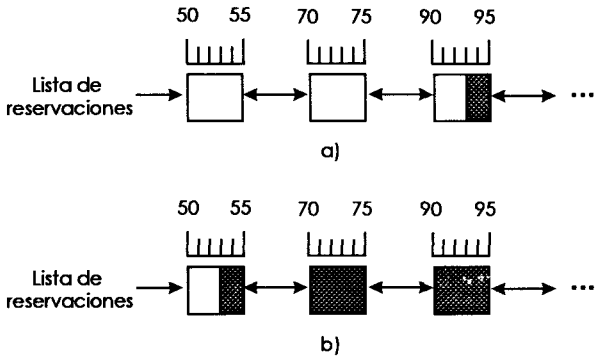


Figura 4.9 Actualización de la lista de reservaciones.
(a) antes de la actualización (b) actualización realizada.

El tiempo t_{un} necesario para detectar el nodo en que debe iniciar la reservación inversa para la nueva tarea es un tiempo constante que implica realizar las operaciones del Algoritmo 4.4, el cual primero calcula el número absoluto de períodos del SPAA que transcurren desde el último cambio en el inicio de su período hasta el plazo de respuesta de la tarea recién llegada T_{n+1} y luego resta a ese valor el número de períodos que ya han transcurrido desde el último cambio en el inicio del período del SPAA hasta el tiempo actual. Como por cada período actual o futuro del SPAA existe un nodo en la lista de reservaciones, el valor obtenido corresponde al número de nodo en que debe iniciar la reservación.

Algoritmo 4.4 Cálculo del nodo en que inicia la reservación.

último posible = plazo de respuesta de T_{n+1} - 1

desfase = último posible - tiempo del último cambio del inicio de período del SPAA

nodo absoluto = desfase / período del SPAA

pasados = tiempo actual - (tiempo del último cambio del inicio de período del SPAA / período del SPAA)

nodo = nodo absoluto - pasados

El tiempo necesario es el tiempo que toma hacer seis operaciones: cuatro restas y dos divisiones. Este tiempo es de $O(1)$.

La prueba de planificabilidad consiste en verificar si el tiempo disponible es suficiente para una nueva reservación. Entonces, para calcular t_{MFC} es necesario revisar la lista de reservaciones. Consideremos el peor de los casos descrito en la Figura 4.8. Para una tarea que requiera reservar todos los intervalos intermedios disponibles y además necesite tiempo hacia los extremos del primer y último intervalos reservados, tendrán que hacerse las operaciones que permitan cuantificar el tiempo disponible tanto en los extremos, como en los $n - 1$ intervalos libres intermedios. Por tanto, el t_{MFC} es de $O(n)$.

El tiempo t_r que toma efectuar la reservación para la nueva tarea T_{n+1} en la lista de reservaciones en el peor de los casos es también de $O(n)$, pues se requiere hacer las operaciones para reservar los $n-1$ intervalos intermedios disponibles y extender las reservaciones hacia los extremos del primer y último intervalos reservados.

Finalmente, el tiempo t_{itc} que tarda la inserción de la nueva tarea en la cola de actividad aperiódica es cuando mucho de $O(n)$, pues la cola está ordenada y en el peor de los casos la tarea debe ser insertada en la última posición.

Entonces, para la llegada de una tarea aperiódica o esporádica t_{ov} es de $O(n)$, ya que este es el orden más grande que presentan sus componentes.

4.8 Soporte de sistema operativo

En [Hull y Liu, 93], [Hull et al., 95] y más recientemente en [Hull et al., 96] se describe una arquitectura enfocada a brindar soporte de sistema operativo para sistemas de tiempo real basados en computación flexible. Sin embargo, el punto central de esta arquitectura es permitir tolerancia a fallas. Para ello, integran un mecanismo de almacenamiento y retorno de resultados intermedios de tareas flexibles con un mecanismo tradicional de puntos de verificación para tolerancia a fallas y recuperación de errores.

La pretensión de esta sección no es proponer una arquitectura particular que soporte la operación del método PTF. Mucho menos enfocar dicha arquitectura hacia el área de tolerancia a fallas, sino identificar los elementos (mecanismos y estructuras) básicos que debe incluir un sistema operativo de tiempo real que implante en su *kernel* el método PTF como algoritmo de planificación.

Los elementos identificados son:

1. Un mecanismo que permita almacenar periódicamente los resultados intermedios de una tarea flexible. Podría ser que el almacenamiento fuera responsabilidad del programador de los algoritmos, pero sería conveniente que el sistema operativo determinara las áreas en que se almacenarán dichos resultados (por ejemplo, añadiendo, para ese propósito, un campo al bloque de control del proceso de la tarea y proveyendo un servicio en tiempo de ejecución que permita a la tarea realizar el almacenamiento). Para aquellas aplicaciones en que los resultados sean visibles en todo momento, como el procesamiento de imágenes y algoritmos como el desarrollado en [Ibargüengoytia, 97] podría no ser necesario el mecanismo de almacenamiento.

2. Un mecanismo para cambiar dinámicamente la prioridad de una tarea flexible.

3. Un mecanismo que permita detectar que una tarea flexible ha llegado a su plazo de respuesta, terminarla y recuperar su último resultado intermedio.

4. Una lista de tareas listas para ejecución en que las tareas presentes estén ordenadas con base en los siguientes criterios:
   - Todas las tareas tienen asignada una prioridad, la cual se utiliza para determinar su posición en la lista.
   - Las subtareas periódicas obligatorias y el SPAA tienen asignada una prioridad determinada con base en la política del algoritmo de razón monotónica.
   - Todas las subtareas periódicas opcionales tienen una misma prioridad, menor que la de cualquier subtarea periódica obligatoria (con ello se garantiza que primero sean ejecutadas todas las subtareas periódicas obligatorias). Las subtareas periódicas opcionales están ordenadas entre sí, con base en un segundo criterio: primero aquella subtarea que al momento haya sido atendida por menos tiempo.
   - Todas las tareas aperiódicas y esporádicas que son atendidas en trasfondo tiene una misma prioridad, menor que la prioridad de las subtareas periódicas opcionales (con ello se garantiza que primero sean ejecutadas todas las subtareas periódicas obligatorias, luego las subtareas periódicas opcionales y finalmente las tareas aperiódicas y esporádicas). Las tareas aperiódicas y esporádicas en trasfondo están ordenadas entre sí, con base en la política del algoritmo EDF.

5. Una cola de actividad aperiódica en la que las tareas aperiódicas y esporádicas presentes estén ordenadas con base en la política del algoritmo EDF.

6. Una lista de reservaciones y sus operaciones asociadas: reservar, actualizar reservación y cancelar reservación.

7. Una operación de manejo de excepciones simple para atender las tareas rechazadas por no cumplir la MFC. Se hace énfasis en que la operación debe ser simple para no impactar en los costos de administración del método.

4.9 Trabajos relacionados a esta tesis

En el capítulo 3 se discutieron los trabajos existentes sobre planificación de tareas flexibles. Sin embargo, se comentó que estos trabajos consideran la planificación de conjuntos de tareas periódicas y aperiódicas de manera independiente. En esta tesis nos valimos de algunos de los resultados de estos trabajos para desarrollar el método PTF.

Poco o nada se ha hecho sobre planificación de conjuntos de tareas periódicas, aperiódicas y esporádicas flexibles. Ello se debe en gran medida a la naturaleza impredecible del comportamiento aperiódico. Lo más próximo a la planificación de conjuntos de tareas periódicas, aperiódicas y esporádicas flexibles encontrado en la literatura es el enfoque de planificación flexible *(flexible scheduling)* [Davis et al., 95], el cual no se ajusta a ninguno de los enfoques principales discutidos en el capítulo 3.

El enfoque de planificación flexible es una estrategia en la que se construye un sistema de tiempo real como un conjunto de tareas críticas, se garantiza su planificabilidad estáticamente y se aprovecha la capacidad de procesamiento aún disponible para ejecutar métodos redundantes más confiables o precisos que mejoren los resultados producidos por las tareas críticas [Davis et al., 95]. El enfoque de planificación flexible tiene varias diferencias respecto al método PTF:

a) El conjunto de tareas a planificar es un subconjunto de tareas periódicas y esporádicas obligatorias, que representan las tareas críticas del sistema que deben ejecutarse dentro de su plazo de respuesta, y un subconjunto de tareas aperiódicas opcionales, que representan métodos más confiables para realizar algunos de los procesos que realizan las tareas obligatorias, pero que requieren más tiempo de ejecución del que se puede garantizar estáticamente. Este conjunto de tareas difiere de las tareas del método PTF en el que se tiene un conjunto de tareas periódicas, aperiódicas y esporádicas compuestas por una parte obligatoria y una parte opcional.
b) Debido a que las tareas obligatorias sólo son periódicas y esporádicas, sus características se conocen a priori y puede hacerse un análisis estático de planificabilidad. Esto no es posible en el método PTF, pues las características de las subtareas obligatorias de las tareas aperiódicas son conocidas hasta el momento de su llegada.

c) Las tareas opcionales no son tareas interrumpibles, sino tareas que deben ejecutarse totalmente antes de cumplir su plazo para mejorar la calidad de los resultados previamente producidos por las tareas obligatorias.

En general, el enfoque de planificación flexible puede atender y asegurar la planificabilidad de las tareas que presentan comportamiento periódico y esporádico, pero no aquellas que presentan comportamiento aperiódico. Por su parte, el método PTF puede atender los tres tipos de tareas, pero con la limitante de que aquellas tareas aperiódicas y esporádicas cuya planificabilidad no pueda garantizarse al momento de su llegada son rechazadas.

4.10 Resumen

En este capítulo se explicó el desarrollo del método PTF y se describió su comportamiento esperado de acuerdo a la capacidad del SPAA y a la actividad aperiódica y esporádica presente. También se presentó un análisis de complejidad del método, haciendo énfasis en el costo de administración producido. Se identificaron los elementos básicos que debe poseer un sistema operativo de tiempo real para soportar la operación del método PTF. Finalmente, se reseñó el enfoque de planificación flexible, un trabajo relacionado al método PTF.

La evaluación práctica del comportamiento del método PTF en diversas circunstancias y su desempeño respecto a otros algoritmos de planificación son aspectos interesantes que fueron evaluados. El siguiente capítulo se encarga de presentar los resultados obtenidos por simulación.

Capítulo 5

RESULTADOS

El capítulo 4 describió el método PTF y la manera de diseñar un servidor periódico de actividad aperiódica. Como parte del diseño del SPAA se identificaron varias situaciones que podrían presentarse dependiendo de la capacidad de procesamiento del SPAA disponible y de la capacidad necesaria de procesamiento del SPAA estimada para la atención de las tareas aperiódicas y esporádicas. Para cada una de estas situaciones se describió el comportamiento esperado del método PTF.

En este capítulo se presentan los resultados obtenidos de las simulaciones realizadas para evaluar si en la práctica el método PTF se ajusta al comportamiento esperado. Además, se compara el método PTF contra cuatro algoritmos de planificación de tareas periódicas, aperiódicas y esporádicas no flexibles. La organización del capítulo es la siguiente.

- La sección 5.1 describe el sistema computacional implantado para simular la operación del método PTF .
- La sección 5.2 presenta los resultados observados por simulación del comportamiento de PTF con diferentes capacidades de procesamiento del SPAA y diferentes estimados de capacidad necesaria para la atención de actividad aperiódica y esporádica. En esta sección

también se presentan los resultados obtenidos al comparar el método PTF contra otros cuatro algoritmos de planificación.

5.1 Ambiente de simulación

Para poder evaluar de manera práctica el comportamiento del método PTF, se diseñó e implantó un ambiente de simulación del método. Dicho ambiente fue programado en lenguaje C y compilado en Borland C++, versión 4.5. El ambiente simulador del método PTF utiliza las librerías OWL *(Object Windows Libraries)* para la construcción de su interfaz en ambiente Windows. Dichas librerías vienen incluidas junto con el compilador utilizado.

Para la comparación del comportamiento del método PTF con respecto a otros algoritmos, se implantaron otros ambientes de simulación para cada uno de ellos. Dichos ambientes fueron escritos para DOS en lenguaje C y compilados en Borland C++ versión 3.1. El hardware utilizado es una PC con procesador Pentium a 133 MHz, con 32 Mbytes de memoria RAM. En la Figura 5.1 se muestra un diagrama de bloques que describe el ambiente de simulación.

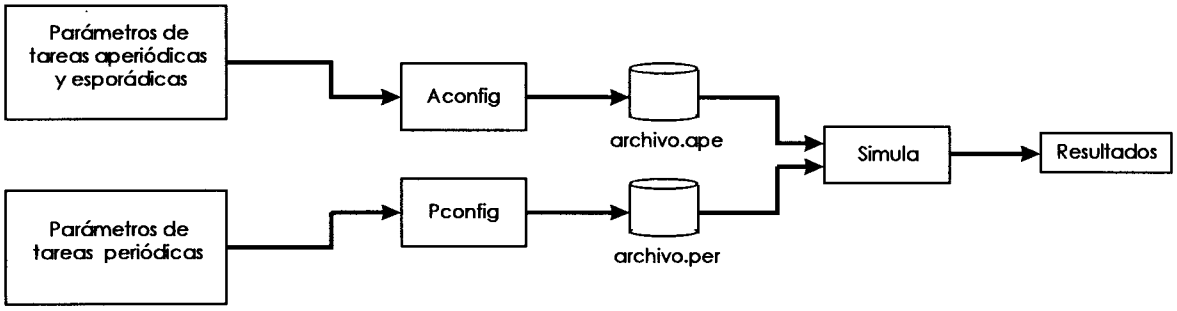


Figura 5.1 Diagrama de bloques del ambiente de simulación.

Los parámetros que caracterizan las tareas aperiódicas y esporádicas son la entrada del programa *Aconfig*. Este programa genera el archivo de configuración *archivo.ape*, el cual sirve como entrada al programa *Simula*. Por otro lado, el programa *Pconfig* recibe los parámetros que caracterizan el conjunto de tareas periódicas y genera el archivo de configuración *archivo.per*. Este archivo también es una entrada del programa *Simula*. El programa *Simula* implanta una simulación del método PTF. Sus resultados son presentados al usuario en pantalla.

Los ambientes de simulación implantados para los algoritmos contra los que se compara el método PTF tienen la misma estructura descrita en la Figura 5.1. La única variante es que el programa *Simula* se sustituye por el programa correspondiente al algoritmo implantado.

5.2 Pruebas del método PTF

5.2.1 Método de pruebas

Para el diseño de las pruebas se detectaron aquellos parámetros que pueden ser objeto de variación de sus valores. Estos parámetros se conocen como variables de entrada. Las variables de entrada detectadas son los siguientes:

- Conjunto de tareas periódicas (incluyendo sus secciones críticas).
- Capacidad del procesamiento y período del SPAA.
- Conjunto de tareas esporádicas.
- Tiempo medio entre arribos de tareas aperiódicas (distribución exponencial).
- Media del tiempo de ejecución de la subtarea obligatoria de una tarea aperiódica (distribución exponencial).
- Media del tiempo de ejecución de la subtarea opcional de una tarea aperiódica (distribución exponencial).
- Plazo de respuesta de las tareas aperiódicas.

En adelante, a un conjunto particular de valores tomados por las variables de entrada se le denominará *configuración*. El elevado número de parámetros de entrada y la gran cantidad de valores que cada uno de ellos puede tomar hacen prácticamente imposible hacer una simulación exhaustiva. Sin embargo, se seleccionaron algunos casos típicos[1], para obtener resultados sobre el

[1] Estos casos son representativos en el sentido de que incluyen las diversas situaciones que pueden presentarse respecto al comportamiento esperado del método PTF definido en la sección 4.6.2.2 y para diferentes cargas de actividad periódica y aperiódica-esporádica.

comportamiento del método PTF. Además, se hicieron las siguientes suposiciones para cuestiones de simulación:

- Cualquier instrucción de una tarea periódica, aperiódica o esporádica tiene un tiempo de ejecución unitario.
- Los costos de administración del método PTF no son considerados.
- El plazo de respuesta de una tarea aperiódica A_i se calcula añadiendo a su tiempo de llegada un tiempo aleatorio (distribución uniforme) en un rango de *2* a *5* veces el tiempo de ejecución total de la tarea (la suma del tiempo de ejecución de sus partes obligatoria y opcional).
- Dos llegadas sucesivas de una tarea esporádica E_i están separadas por un tiempo igual a la suma de su mínimo tiempo entre arribos mta_i y un tiempo aleatorio (distribución uniforme) en un rango de 0 a 100 unidades.
- Las tareas aperiódicas y esporádicas rechazadas producen un error unitario (error = 1).

Debido a que algunos parámetros de entrada no son fijos sino que siguen ciertas distribuciones de probabilidad, cada prueba fue repetida 50 veces. Los resultados producidos por cada repetición fueron registrados y promediados. Los promedios obtenidos son los resultados presentados en las siguientes secciones.

5.2.2 Pruebas sobre comportamiento esperado del método PTF

En el capítulo 4, sección 4.6.2.2, se hizo una descripción del comportamiento esperado del método PTF para diferentes capacidades de procesamiento del SPAA (τ_o y τ_o') y de los estimados de capacidad de procesamiento necesaria del SPAA *(Q)* y de capacidad de procesamiento mínima aceptable del SPAA *(Q')* para la atención de tareas aperiódicas y esporádicas. En esta sección se describen los resultados de las pruebas realizadas para determinar si en la práctica el método PTF se ajusta a dicho comportamiento esperado. Para ello, se diseñaron 13 pruebas que involucran todas las posibles situaciones descritas en el capítulo 4. El diseño de cada prueba se describe en el apéndice A. La Tabla 5.1 muestra los resultados obtenidos para las diferentes pruebas.

Cuando en la columna denominada *Capacidad del SPAA* se encuentra un campo del tipo $\tau_o = x$, debe entenderse que la capacidad del servidor periódico es la máxima posible que garantiza la planificabilidad de las tareas periódicas completas, esto es, las partes opcionales se consideran como si fuesen obligatorias. Si el campo es de la forma $\tau_o' = x$, la capacidad del servidor periódico es la máxima posible que garantiza la planificabilidad de las partes obligatorias del conjunto de tareas periódicas en cuestión. Si el campo no toma ninguna de las dos formas, la capacidad del servidor periódico no corresponde a ninguno de los casos anteriores, pero es menor a τ_o'.

Tabla 5.1 Resultados de la simulación para evaluar el comportamiento del método PTF en diversas situaciones.

Prueba	*Capacidad del SPAA*	*Q*	*Q'*	*% error aperiódicas y esporádicas (EAE)*	*% error periódicas (EP)*	*% aperiódicas y esporádicas rechazadas (AER)*	*% de utilización del CPU*
1	$\tau_o = 7$	4.98	-	**4.93**	**0.00**	3.69	61.28
2	$\tau_o = 7$	15.55	3.10	**8.65**	**0.00**	6.46	71.76
3	$\tau_o = 7$	-	8.49	**24.48**	**0.00**	20.45	77.77
4	$\tau_o' = 14$	-	8.49	**11.65**	**1.98**	6.79	82.12
5	$\tau_o = 6$	4.06	-	**4.53**	**0.00**	2.63	57.24
6	$\tau_o = 6$	10.15	2.69	**20.68**	**0.00**	15.3	74.7
7	$\tau_o = 6$	-	10.56	**37.21**	**0.00**	31.55	83.46
8	$\tau_o' = 12$	9.47	-	**4.81**	**1.11**	2.53	81.39
9	$\tau_o' = 6$	2.79	-	**4.65**	**5.45**	2.07	94.56
10a	$\tau_o' = 14$	28.68	5.27	**27.95**	**37.29**	6.31	99.45
10b	12	28.68	5.27	**36.16**	**30.46**	8.99	99.52
10c	10	28.68	5.27	**45.15**	**24.00**	11.95	99.57
10d	8	28.68	5.27	**55.55**	**16.30**	16.36	99.64
10e	6	28.68	5.27	**64.02**	**10.88**	21.88	99.60
11a	$\tau_o' = 6$	10.97	2.83	**10.23**	**0.26**	6.03	81.93
11b	5	10.97	2.83	**12.40**	**0.07**	7.79	81.80
11c	4	10.97	2.83	**14.92**	**0.02**	9.80	81.38
11d	3	10.97	2.83	**18.10**	**0.00**	12.93	81.00
12	$\tau_o' = 9$	-	9.90	**32.77**	**2.77**	23.66	92.85
13	$\tau_o' = 6$	-	7.39	**32.77**	**11.21**	18.33	94.99

Q = capacidad de procesamiento necesaria del SPAA.

Q' = capacidad de procesamiento mínima aceptable del SPAA.

Algunos campos de las columnas Q y Q' se encuentran vacíos. Esto no quiere decir que su valor no pueda ser calculado, sino que para la prueba en cuestión dicho valor no es de interés.

Los porcentajes de error presentados en la Tabla 5.1 están calculados con base en el error máximo posible que podría presentarse tanto en las tareas periódicas como en las aperiódicas y esporádicas, es decir, cuando todas las tareas producen un error de valor unitario (error = *1*). Por ejemplo, el error máximo posible de las tareas aperiódicas y esporádicas se presenta cuando todas las tareas aperiódicas y esporádicas producen un error de valor unitario.

La Fórmula (5.1) muestra la manera en que se calcula el porcentaje de error de las tareas periódicas. Recuérdese que E es el error promedio de las tareas periódicas, el cual es un valor entre *0* y *1* que se calcula mediante la Fórmula (3.5) (sección 3.3.3.2.1).

$$\%\ error\ periódicas = E \bullet (100) \tag{5.1}$$

La Fórmula (5.2) muestra la manera en que se calcula el porcentaje de error de las tareas aperiódicas y esporádicas. ε es el error total de las tareas aperiódicas y esporádicas calculado con la Fórmula (3.2) (sección 3.3.1).

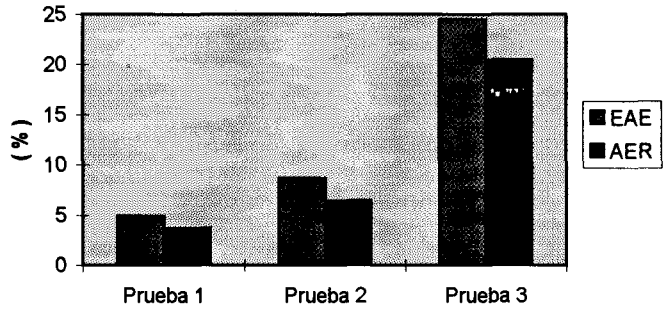
$$\%\ error\ aperiódicas\ y\ esporádicas = \frac{error\ máximo\ posible\ aperiódicas\ y\ esporádicas}{\varepsilon}(100) \tag{5.2}$$

Pruebas 1 a 7 - Garantizando la planificabilidad de las tareas aperiódicas completas

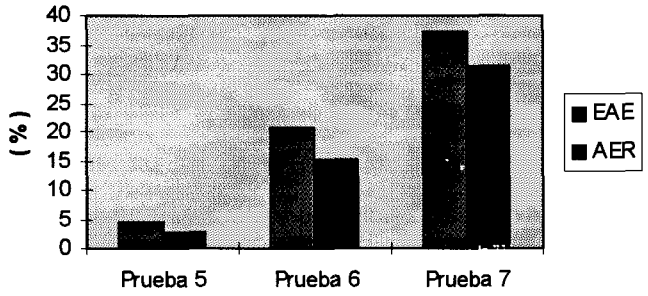
Las pruebas 1, 2 y 3 constituyen una serie de pruebas en que para el mismo conjunto de tareas periódicas, se varían los requerimientos de procesamiento de las tareas aperiódicas y esporádicas, de modo que se establecen diferentes relaciones entre la capacidad del servidor periódico τ_o y los estimados Q y Q'. Lo mismo ocurre con las pruebas 5, 6 y 7.

Las pruebas 1 y 5 representan el caso en que $\tau_o > Q$. De los resultados de la Tabla 5.1 y las Gráficas relacionadas 5.1 y 5.2, puede observarse que los niveles de error de las tareas aperiódicas y esporádicas son muy bajos y que gran parte de ese error es producido por las tareas rechazadas por no cumplir la MFC (recordemos que para cuestiones de simulación se asumió que las tareas rechazadas producen un error de *1*). La planificabilidad de las tareas periódicas completas está garantizada en ambos casos; el conjunto de tareas periódicas (incluyendo sus partes obligatoria y

opcional) es planificable sin error junto con un servidor periódico de actividad aperiódica con parámetros τ_o y π_o según la ecuación (4.8).



Gráfica 5.1 Pruebas 1, 2 y 3.
(EAE = Porcentaje de error de las tareas aperiódicas y esporádicas
AER = Porcentaje de aperiódicas y esporádicas rechazadas)



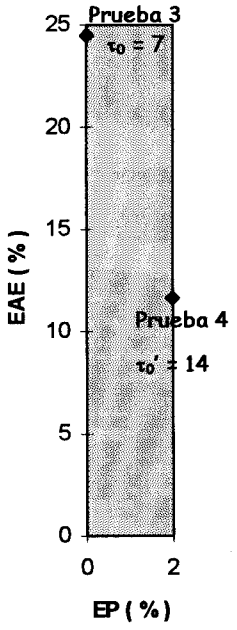
Gráfica 5.2 Pruebas 5, 6 y 7.

Las pruebas 2 y 6 representan el caso en que $Q' \leq \tau_o \leq Q$. En dichas pruebas se observa que se incrementa el número de tareas rechazadas y el porcentaje de error de las tareas aperiódicas y esporádicas respecto a las pruebas 1 y 5. Sin embargo, estos errores aún son pequeños. A pesar de que los valores de τ_o, Q y Q' guardan cierta similitud entre ambas pruebas, el incremento en el error producido en las tareas aperiódicas y esporádicas en el Prueba 6 fue sustancialmente mayor al producido en la Prueba 2. Para tratar de explicar este fenómeno se acudió a la configuración de cada una de las pruebas. En el caso de la Prueba 2 se encontró que el tiempo medio entre arribos es de *100*, casi dos y media veces el período del SPAA, mientras que en el caso de la Prueba 6 el tiempo medio entre arribos es de *20*, un poco más de dos tercios del período del SPAA. Esto implica que es más probable que se presente una ráfaga de tareas aperiódicas en el caso de la Prueba 6 que en el

caso de la Prueba 2. Además, debido a que el tiempo medio entre arribos es menor que el período del SPAA, es muy probable que no sea posible reservar tiempo de procesamiento en la lista de reservaciones para todas las tareas que llegan en ráfaga. De esta manera, algunas de ellas son rechazadas.

Cuando la capacidad del servidor periódico τ_o es menor al estimado de la capacidad de procesamiento mínima aceptable del SPAA Q', se espera que se produzca un error considerable en la atención de las tareas aperiódicas y esporádicas. Las pruebas 3 y 7 se diseñaron para evaluar este caso.

Se observa que el porcentaje de error producido se incrementó sustancialmente respecto a las pruebas 2 y 6 (283 % y 179 %, respectivamente). Además, se aprecia que mientras por un lado, las tareas periódicas se planifican sin error, las tareas aperiódicas comienzan a tener errores considerables. Esto no es deseable, sería preferible establecer un compromiso entre los errores de las tareas aperiódicas y esporádicas y los de las tareas periódicas.



Gráfica 5.3 Pruebas 3 y 4 (EP = Porcentaje de error de las tareas periódicas)

Para las pruebas 1, 2 y 3 se tienen tareas periódicas con partes obligatorias y opcionales, no así en el caso de las pruebas 5, 6 y 7 en que las tareas periódicas no tienen parte opcional. Buscando un compromiso entre el error total de las tareas aperiódicas y esporádicas y el error promedio de las tareas periódicas se decidió diseñar una nueva prueba, la Prueba 4, a partir de la Prueba 3. En ella se asegura la planificabilidad solamente para la parte obligatoria de las tareas periódicas, permitiendo crear un SPAA con una mayor capacidad de procesamiento para la atención de las tareas aperiódicas y esporádicas.

De los resultados de las pruebas 3 y 4 (Tabla 5.1 y Gráfica 5.3) se observa que el hecho de garantizar la planificabilidad sólo para la parte obligatoria de las tareas periódicas, permitiendo así un SPAA con mayor capacidad de procesamiento, puede producir un mejoramiento drástico en el error de las tareas aperiódicas y esporádicas mientras provoca un mínimo perjuicio en el error de las tareas periódicas. Por lo tanto se deduce que es preferible

considerar sólo la parte obligatoria de las tareas periódicas para determinar la capacidad de procesamiento del servidor periódico de actividad aperiódica. Dos situaciones importantes se obtienen de determinar la capacidad del servidor periódico de esta manera:

- Se brinda una mejor servicio a las tareas aperiódicas y esporádicas, disminuyendo su error.
- Se continúa garantizando que la parte obligatoria de ninguna tarea periódica perderá su plazo de respuesta.

Pruebas 8 y 9 - $\tau_o' > Q$

Las pruebas 8 y 9 presentan el caso en que $\tau_o' > Q$. En la Prueba 9, el conjunto de tareas periódicas completas no pasa la prueba de planificabilidad dada por la ecuación (4.8), sin embargo sí se garantiza la planificabilidad de sus partes obligatorias.

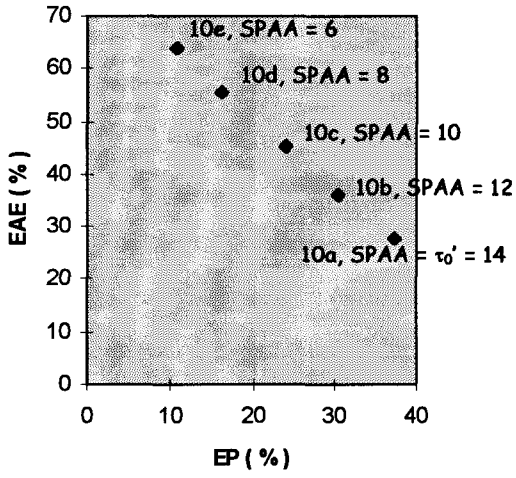
En general se observan niveles de error muy bajos tanto para las tareas periódicas como para las tareas aperiódicas y esporádicas. Estos resultados nos indican que el hecho de explotar la capacidad de procesamiento máxima posible del SPAA τ_o', cuando $\tau_o' > Q$, permite mejorar la atención de las tareas periódicas y aperiódicas sin afectar considerablemente el servicio de las tareas periódicas.

Pruebas 10*x* y 11*x* - Buscando la capacidad del SPAA más conveniente.

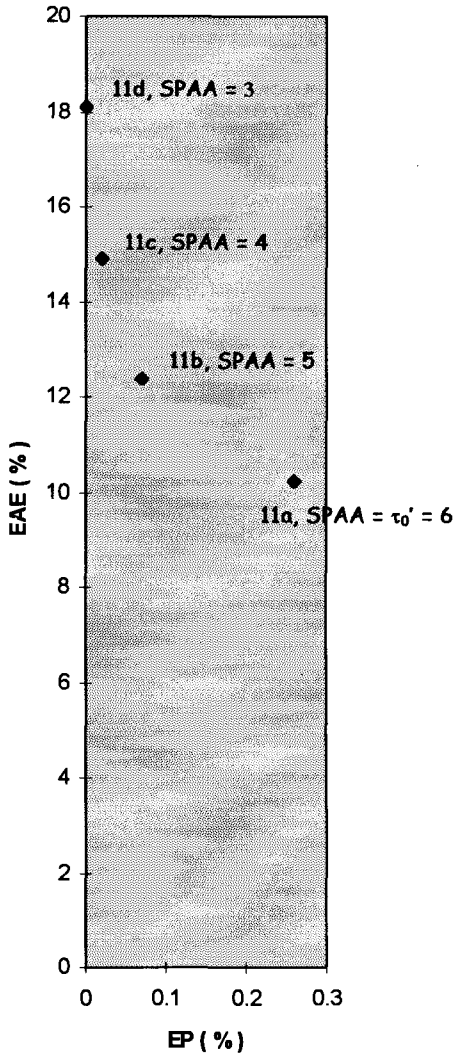
Estas pruebas resultan ser los más interesante ya que permitieron observar el comportamiento de los niveles de error producidos por las tareas periódicas y por las tareas aperiódicas y esporádicas con diferentes capacidades de procesamiento del servidor periódico.

Para el diseño de las pruebas se seleccionó un caso en el que la utilización total del procesador ($U = \tau_1/\pi_1 + \tau_2/\pi_2 + \ldots + \tau_v/\pi_n$) por parte del conjunto de tareas periódicas es alta (Prueba 10*x*, $U_{10} = 0.9$) y otro en el que dicha utilización es regular (Prueba 11*x*, $U_{11} = 0.6786$). De hecho, las tareas periódicas completas de la Prueba 10*x* (incluyendo las partes opcionales) no serían planificables, ya que no cumplen la prueba de planificabilidad dada por la ecuación (4.8). Por otro lado, la utilización del procesador de la partes opcionales ($U_{op} = o_1/\pi_1 + o_2/\pi_2 + \ldots + o_n/\pi_n$) de las tareas periódicas es grande respecto a la total en el caso de la Prueba 10*x* ($U_{10op} = 0.5217$) y pequeña en el caso de la Prueba 11*x* ($U_{11op} = 0.1655$).

La simulación consistió en ir disminuyendo la capacidad del SPAA desde la máxima posible (τ_o') hasta el entero mayor más próximo a Q' y observar el comportamiento de los errores producidos tanto para tareas periódicas como para aperiódicas y esporádicas. En ambos casos $Q' \leq \tau_0' \leq Q$.



Gráfica 5.4 Prueba 10*x*



Gráfica 5.5 Prueba 11*x*

Primeramente se observa que cuando la utilización del procesador de las partes opcionales de las tareas periódicas U_{op} es elevado (prueba 10*x*), el error producido por las tareas periódicas puede resultar importante. Esto es debido a que las necesidades de procesamiento de estas partes opcionales podrían llegar a traspasar ampliamente la capacidad de ejecución disponible en trasfondo una vez que se planifica un SPAA con capacidad de procesamiento τ_o' junto con la parte obligatoria de las tareas periódicas. Aunque la observación se hizo para el caso en que $Q' \leq \tau_0' \leq Q$, el caso en que τ_0 ' > Q no está exento de presentar el mismo fenómeno, con lo cual la conclusión obtenida de las pruebas 8 y 9 no es totalmente cierta.

Si los requerimientos de procesamiento de las partes opcionales de las tareas aperiódicas son elevados, podría ser que τ_o' no sea el valor más apropiado para la capacidad del SPAA. Por ejemplo, de los resultados de la Prueba 10, incisos (a), (b), (c), (d) y (e), presentados en la Tabla 5.1 y la Gráfica 5.4, se intuye que una capacidad de *13* produciría porcentajes de error más equitativos entre las tareas periódicas y las aperiódicas y esporádicas. Una herramienta de simulación como la utilizada para la obtención de estos resultados es de gran ayuda en la determinación de la capacidad más conveniente.

En cambio, cuando la utilización del procesador de las partes opcionales es pequeña es probable que el tiempo de ejecución en trasfondo sea suficiente para producir errores insignificantes para las tareas periódicas, como se aprecia en los resultados de la Prueba 11a. En este caso, no es conveniente una disminución de la capacidad del servidor periódico, ya que el mejoramiento de la atención a las tareas periódicas es insignificante respecto al perjuicio que se hace a la atención de las tareas aperiódicas y esporádicas, como se observa en la Tabla 5.1 y la Gráfica 5.5.

Pruebas 12 y 13 - $\tau_o' \leq Q'$

El interés de evaluar este caso es determinar qué tan buena resulta la comparación $\tau_o' \leq Q'$ como criterio para determinar que es preferible buscar equipos con mayor capacidad de procesamiento.

Se diseñaron las pruebas 12 y 13 para evaluar este caso. En ambos, el valor de τ_o' es menor, pero muy cercano a Q'. Los resultados de las pruebas pueden observarse en la Tabla 5.1. Coincidentemente, el porcentaje de error de las tareas aperiódicas y esporádicas es idéntico para ambas pruebas, lo cual no es lo que se espera para todos los casos. Aunque este error parece aún pequeño, es evidente que entre mayor sea la diferencia entre τ_o' y Q' mayor será el error producido. Por lo tanto, $\tau_o' \leq Q'$ más que un criterio para decidir utilizar otros procesadores, es un punto de alerta a partir del cual se debe evaluar si el error producido por las tareas aperiódicas y esporádicas aún es aceptable o es mejor buscar un equipo con mayor capacidad de procesamiento en el cual se produzcan mejores resultados.

5.2.2.1 Conclusiones obtenidas de la simulación.

En general, el comportamiento del método PTF no se ajusta del todo con lo que se describió en el capítulo 4 como comportamiento esperado (sección 4.6.2.2). Esto se debe principalmente a que se

esperaba que fuera práctico planificar las tareas periódicas completas, pero se observó que ello afecta considerablemente la atención que podría darse a las tareas aperiódicas y esporádicas.

En particular, de la simulación realizada se obtuvieron las siguientes conclusiones respecto al comportamiento del método PTF:

1. Siempre es preferible determinar la capacidad de procesamiento del SPAA considerando sólo la parte obligatoria de las tareas periódicas, esto asegura niveles de error menores para las tareas aperiódicas y esporádicas. Con ello, los incisos (a), (b), (c) y (d) del comportamiento esperado del método PTF (sección 4.6.2.2) son imprácticos y no deben ser tomados en cuenta.
2. Cuando la utilización del procesador de las partes opcionales de las tareas periódicas es elevada, es probable que la capacidad de procesamiento en trasfondo no sea suficiente para darles una buena atención, luego de que se planifican juntos un SPAA con capacidad de procesamiento τ_o' y las partes obligatorias del conjunto de tareas periódicas. Entonces, podrían producirse niveles de error de las tareas periódicas muy importantes, mientras que las tareas aperiódicas y esporádicas producen niveles de error muy bajos. Si este es el caso, vale la pena invertir algo de tiempo en realizar una simulación para determinar si existe una capacidad de procesamiento menor a τ_o' que produzca un error más equitativo entre las tareas periódicas y las aperiódicas y esporádicas.
3. Q es un buen estimado para la capacidad de procesamiento necesaria para la atención de las tareas aperiódicas y esporádicas (ver sección 4.6.2.1, página 83). En las pruebas 1, 5, 8 y 9 se observa que cuando la capacidad del SPAA es mayor a Q, el porcentaje de error de las tareas aperiódicas y esporádicas es muy bajo.
4. Más que un criterio para decidir utilizar otros procesadores, $\tau_o' \leq Q'$ es un punto de alerta a partir del cual se debe evaluar si el error producido por las tareas aperiódicas y esporádicas es aún aceptable o es mejor buscar un equipo con mayor capacidad de procesamiento en el cual se produzcan mejores resultados.

5.2.3 Pruebas del método PTF contra otros algoritmos

En esta sección se presentan los resultados obtenidos por simulación al planificar un conjunto de tareas periódicas, aperiódicas y esporádicas flexible por el método PTF y otros cuatro algoritmos. El

objetivo es determinar en qué situaciones es preferible utilizar el método PTF a utilizar alguno de los otros algoritmos de planificación. En la sección 5.2.3.1 se introducen los cuatro algoritmos contra los que se compara el método PTF. La sección 5.2.3.2 presenta los casos evaluados y los resultados obtenidos. El diseño de las pruebas se describe en el apéndice A.

5.2.3.1 Algoritmos de planificación comparados contra el método PTF

Se eligieron cuatro algoritmos de planificación de tareas periódicas, aperiódicas y esporádicas contra los cuales se comparó el método PTF. Todos ellos son algoritmos no flexibles (no hacen distinción entre las partes obligatoria y opcional de las tareas), ya que, como se manifestó en el capítulo 4, los algoritmos de planificación de tareas flexibles existentes no consideran la combinación de actividad periódica, aperiódica y esporádica.

La comparación se realizó con base en los errores producidos por cada algoritmo y el método PTF al ser ejecutados bajo la misma configuración. Cada prueba fue repetida 50 veces, por algoritmo, debido a las condiciones de aleatoreidad que presentan las tareas aperiódicas y esporádicas.

A continuación se describen los algoritmos contra los cuales se comparó el método PTF.

a) *Atención en trasfondo (background).* Las tareas periódicas se planifican de acuerdo a la política del algoritmo de razón monotónica. La prioridad de las tareas aperiódicas y esporádicas es menor que la prioridad más baja de cualquier tarea periódica, por lo que sólo son ejecutadas cuando no hay tareas periódicas listas para ejecución. Las tareas aperiódicas y esporádicas se atienden de acuerdo a la política FIFO (*First In, First Out*) [Lehoczky et al., 91].

b) *Atención en trasfondo mejorado.* Este algoritmo es una variante del algoritmo anterior que brinda atención a las tareas aperiódicas y esporádicas con base en la proximidad de su plazo de respuesta y no con base en su orden de llegada. Con ello se asegura que primero sea atendida la tarea más urgente. Los dos cambios importantes respecto al algoritmo de atención en trasfondo son:
    - Las tareas aperiódicas y esporádicas se atienden con base en la política EDF.

- Cualquier tarea aún no completada es terminada por el algoritmo cuando se llega su plazo de respuesta.

c) *Servidor diferido (deferreable server).* Un servidor diferido (SD) es una tarea con período π_{SD} y capacidad de procesamiento τ_{SD}, la cual es planificada junto con otras tareas periódicas por el algoritmo de razón monotónica. El SD se usa para brindar atención con alta prioridad a las tareas aperiódicas (normalmente su período es menor al de cualquier otra tarea periódica), de modo que está disponible para atender tareas aperiódicas desde el inicio de su período, hasta que agota su capacidad de procesamiento o alcanza el final de su período. La capacidad de procesamiento τ_{SD} del SD está disponible a lo largo de todo su período, y aquella que no es utilizada, se pierde cuando éste termina. En ese momento se restaura la capacidad de procesamiento total τ_{SD}. Las tareas aperiódicas que llegan cuando la capacidad de procesamiento está agotada son atendidas en trasfondo. La capacidad del SD es la máxima posible que mantiene planificable el conjunto de tareas periódicas y el servidor diferido. La atención de las tareas aperiódicas se hace con base en la política FIFO [Strosnider et al., 95].

En [Strosnider et al., 95] se propone que se implante un servidor diferido por cada tarea esporádica, o bien, que en un solo SD se reserve una parte de la capacidad de procesamiento total para uso exclusivo de las tareas esporádicas. Esto es porque en general las tareas aperiódicas tienen plazos de respuesta flexibles y las esporádicas tienen plazos de respuesta estrictos. Sin embargo, como se discutió en el capítulo 4, el plazo de las tareas aperiódicas y esporádicas flexibles es igualmente importante, por lo que para la simulación se utiliza un solo SD con la capacidad de procesamiento máxima posible, sin privilegiar la atención de las tareas esporádicas.

d) *Servidor diferido mejorado.* Este algoritmo presenta dos cambios respecto al algoritmo anterior:

- Las tareas aperiódicas y esporádicas se atienden con base en la política EDF.
- Cualquier tarea aún no completada es terminada por el algoritmo cuando se llega su plazo de respuesta.

El error producido por las tareas es el criterio de comparación de estos algoritmos contra el método PTF. A continuación se reseñan las consideraciones hechas para el cálculo del error en dichos algoritmos.

En los algoritmos (a) y (c) todas las tareas se ejecutan hasta su término. Para estos casos, una tarea que termina habiendo perdido su plazo de respuesta produce un error de uno. En el caso de los algoritmos (b) y (d) una tarea no completada puede ser terminada al llegarse su plazo. Aunque se especificó que los algoritmos descritos no son flexibles, y estrictamente un plazo perdido representa un error de uno, les otorgaremos cierto grado de flexibilidad, haciendo la suposición de que una tarea que pierde su plazo de respuesta devuelve como resultado el valor calculado hasta ese momento, de modo que los errores de las tareas pueden calcularse con las funciones de error de las fórmulas (3.1) y (3.3); las mismas utilizadas para el cálculo de errores para el método PTF.

5.2.3.2 Simulación.

Para evaluar en diferentes circunstancias el comportamiento del método PTF y de los algoritmos descritos en la sección 5.2.3.1, se diseñaron seis pruebas, detalladas en el apéndice A (la mayoría de ellas son variantes de las pruebas presentadas previamente).

Sea L el miembro izquierdo de la ecuación (4.9):

$$\max_{1\le k\le n} \min_{t\in H_k} \left\{ \frac{1}{t} \left(\sum_{j=1}^{k-1} \tau_j \left\lceil \frac{t}{\pi_j} \right\rceil + \tau_k + B_k \right) \right\} \le 1,$$

El valor de L permite determinar la planificabilidad de un conjunto de tareas periódicas considerando el bloqueo producido por la compartición de recursos. Digamos que hay una *carga periódica baja* cuando $L \le 0.5$, una *carga periódica media* cuando $0.5 < L \le 0.9$, y una *carga periódica alta* cuando $L > 0.9$. El valor de L se calcula considerando las tareas periódicas completas (incluyendo las partes obligatoria y opcional). De la ecuación (4.9) recordemos que si $L > 1$, las tareas periódicas completas no son planificables, pero es probable que sí lo sean las partes obligatorias de las tareas periódicas.

Definamos como *actividad aperiódica y esporádica moderada* al caso en que $Q < \tau_o'$, y como *actividad aperiódica y esporádica intensa* al caso en que $Q' \leq \tau_o' \leq Q$. Las pruebas diseñadas para evaluar el comportamiento del método PTF y los otros algoritmos comprenden los siguientes casos:

a) *Prueba 14.* Carga periódica baja y actividad aperiódica y esporádica moderada.
b) *Prueba 15.* Carga periódica baja y actividad aperiódica y esporádica intensa.
c) *Prueba 16.* Carga periódica media y actividad aperiódica y esporádica moderada.
d) *Prueba 17.* Carga periódica media y actividad aperiódica y esporádica intensa.
e) *Prueba 18.* Carga periódica alta y actividad aperiódica y esporádica moderada.
f) *Prueba 19.* Carga periódica alta y actividad aperiódica y esporádica intensa.

Es importante hacer notar que para los casos en que se presenta actividad aperiódica y esporádica intensa se hizo la consideración de que el tamaño de la partes opcionales de una tarea aperiódica o esporádica es por lo menos dos veces el tamaño de su parte obligatoria.

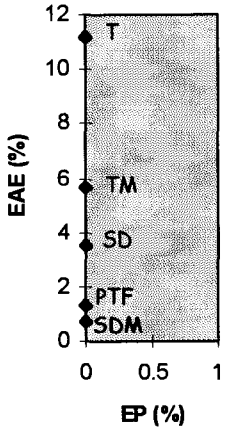
Los resultados obtenidos por simulación de los seis casos se presentan en la Tabla 5.2.

Tabla 5.2 Resultados obtenidos de la ejecución de los cinco algoritmos comparados.

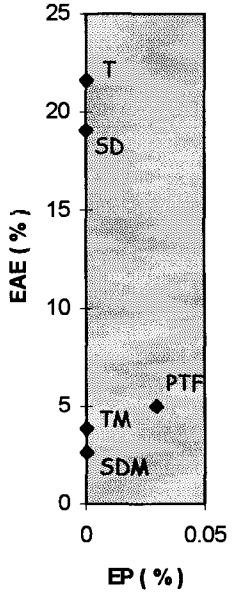
Prueba	*L*	*T*		*TM*		*SD*		*SDM*		*PTF*	
		EAE	*EP*	*EAE*	*EP*	*EAE*	*EP*	*EAE*	*EP*	*EAE*	*EP*
14	0.3000	11.22	0.00	5.65	0.00	3.52	0.00	**0.76**	**0.00**	1.32	0.00
15	0.3000	21.63	0.00	3.92	0.00	19.07	0.00	**2.65**	**0.00**	5.05	0.03
16	0.6750	20.14	0.00	17.84	0.00	3.70	0.00	2.90	0.00	**0.34**	**0.01**
17	0.6750	18.85	0.00	9.38	0.00	13.27	0.00	4.93	0.00	**3.03**	**0.66**
18	1.0571	62.88	0.00	56.18	1.90	62.70	7.79	56.78	1.90	**4.57**	**5.44**
19	1.0750	100.00	3.40	79.95	2.55	100.00	3.40	80.47	2.55	**27.40**	**37.25**

T = Atención en trasfondo
TM = Atención en trasfondo mejorado
SD = Servidor diferido
SDM = Servidor diferido mejorado
PTF = Método de planificación de tareas periódicas, aperiódicas y esporádicas flexibles.
EAE = Porcentaje de error de las tareas aperiódicas y esporádicas
EP = Porcentaje de error de las tareas periódicas

En la Tabla 5.2 se observa que prácticamente en todos los casos el método PTF y el servidor diferido mejorado superan a los algoritmos de atención en trasfondo, atención en trasfondo mejorado y servidor diferido. Para los casos en que la carga periódica es baja, el algoritmo de servidor diferido mejorado tuvo un mejor comportamiento que el método PTF (ver Gráficas 5.6 y 5.7). Esto es previsible, ya que la capacidad de procesamiento en trasfondo del algoritmo de servidor diferido mejorado es muy amplia y permite que prácticamente todas las tareas aperiódicas y esporádicas sean ejecutadas por completo. Por su parte PTF sólo puede administrar la capacidad de procesamiento del SPAA y aunque existe una capacidad de procesamiento en trasfondo importante, PTF no puede aprovecharla en favor de las tareas aperiódicas y esporádicas ocasionando el rechazo de algunas tareas que podrían ser planificadas sin problema alguno.



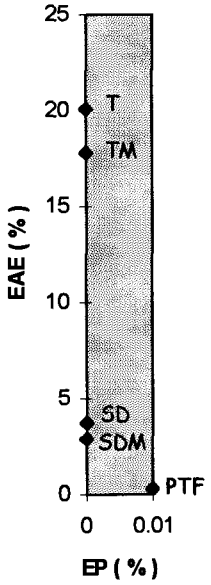
Gráfica 5.6 Prueba 14.
Carga periódica baja y actividad aperiódica y esporádica moderada



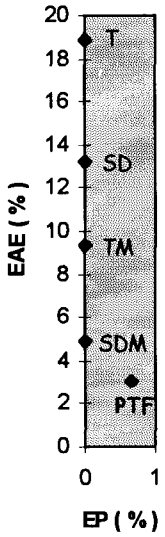
Gráfica 5.7 Prueba 15.
Carga periódica baja y actividad aperiódica y esporádica intensa

Cuando la carga periódica es media, se observa que PTF supera ligeramente al algoritmo de servidor diferido mejorado (ver Gráficas 5.8 y 5.9). Aunque podría asumirse que el comportamiento de ambos algoritmos es prácticamente el mismo, el uso del método PTF garantiza que ninguna tarea aperiódica aceptada pierda su plazo de respuesta, lo cual no sucede con el otro algoritmo. Además, aquellas tareas que son rechazadas por PTF pueden ser atendidas por una operación de manejo de excepciones, la cual aunque quizá sea muy simple por las limitaciones de tiempo, es preferible a los

efectos no controlados producidos por la pérdida de un plazo. Desafortunadamente, el método PTF es más complejo que el algoritmo de servidor diferido modificado, por lo que sus costos de administración son más elevados. Dichos costos no fueron considerados en las simulaciones realizadas. Sin embargo, se espera que en una implantación real, el desempeño del método PTF se vea afectado por tales costos de administración.



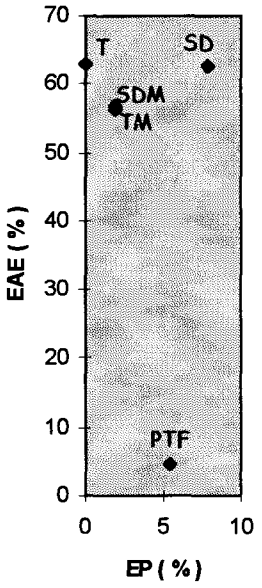
Gráfica 5.8 Prueba 16.
Carga periódica media y
actividad aperiódica y esporádica
moderada



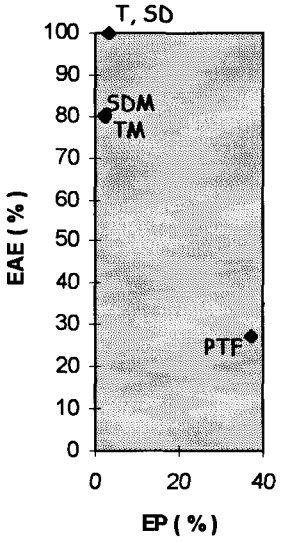
Gráfica 5.9 Prueba 17.
Carga periódica media y
actividad aperiódica y esporádica
intensa

El mejor comportamiento del método PTF respecto a los otros algoritmos se presenta cuando la carga periódica es alta. Mientras el método PTF produce errores más equitativos entre las tarea periódicas y las aperiódicas y esporádicas, como se observa en la Tabla 5.2 y las Gráficas 5.10 y 5.11, los otros métodos producen errores grandes en la ejecución las tareas aperiódicas y esporádicas (principalmente producidos por la pérdida impredecible de plazos), y pequeños en la atención de las tareas periódicas. Esto es explicable, ya que mientras más alta sea la carga periódica, menor será la capacidad de procesamiento de los servidores diferidos y la capacidad de procesamiento en trasfondo. Entonces, mientras que las tareas periódicas se ejecutan con un nivel de error bajo, las tareas aperiódicas y esporádicas sufren una atención muy pobre.

Inclusive, en las pruebas 18 y 19 se observa que L tiene un valor que indica que las tareas periódicas completas no son planificables. Esto ocasiona que no pueda crearse un servidor diferido y que los algoritmos de servidor diferido y servidor diferido mejorado se conviertan en atención en trasfondo y atención en trasfondo mejorado, respectivamente. Con este valor de L, el algoritmo de servidor diferido mejorado no garantiza siquiera el cumplimiento de los plazos de respuesta de las tareas periódicas.



Gráfica 5.10 Prueba 18.
Carga periódica alta y actividad aperiódica y esporádica moderada



Gráfica 5.11 Prueba 19.
Carga periódica alta y actividad aperiódica y esporádica intensa

En conclusión, el método PTF es una técnica de planificación que supera a otros algoritmos conocidos en situaciones de sobrecarga, en las que el tiempo disponible no es suficiente para cumplir los requerimientos temporales impuestos por las tareas. En estas condiciones, PTF garantiza el cumplimiento de los plazos de todas las tareas periódicas y de todas las tareas aperiódicas y esporádicas que cumplen la restricción de viabilidad de la parte obligatoria. Sin embargo, en situaciones en que los requerimientos de procesamiento son moderados, como en los casos de carga periódica baja, tiene la desventaja de que no es capaz de aprovechar adecuadamente la capacidad de procesamiento en trasfondo y podría rechazar tareas que podrían ser perfectamente planificables, a diferencia del algoritmo de servidor diferido modificado.

5.3 Resumen

En este capítulo se presentó la evaluación práctica de las ideas teóricas desarrolladas en esta tesis. Primeramente se describió el ambiente utilizado para desarrollar la etapa de simulación. Posteriormente, se describieron las pruebas diseñados para evaluar si el comportamiento esperado del método PTF se presentaba en la práctica. Se observó que algunos casos descritos en el comportamiento esperado eran imprácticos y se obtuvieron conclusiones sobre el comportamiento real del método PTF. Finalmente, se comparó el método PTF contra otros cuatro algoritmos con base en los errores producidos tanto para las tareas periódicas, como para las aperiódicas y esporádicas. Se concluyó que PTF es un método de planificación que, en situaciones de sobrecarga, produce niveles de error menores en la ejecución de las tareas flexibles que los algoritmos contra los que fue comparado. Sin embargo, en situaciones de requerimientos de procesamiento moderados, es preferible utilizar alguno de los otros algoritmos, ya que sus costos de administración son más bajos.

Capítulo 6

CONCLUSIONES

6.1 Conclusiones

En este trabajo de tesis se logró desarrollar un método de planificación de tareas periódicas, aperiódicas y esporádicas flexibles para un sistema uniprocesador que permite mantener la estabilidad del sistema en situaciones de sobrecarga. Este trabajo es una aportación al campo de investigación sobre computación flexible y sistemas de tiempo real, y está motivado en gran parte por el interés de ofrecer un método de planificación para conjuntos de tareas periódicas, aperiódicas y esporádicas flexibles, ya que, en la práctica, los sistemas de tiempo real se componen de mezclas de los tres tipos de tareas y los algoritmos existentes sólo consideran la planificación de actividad periódica y aperiódica de manera independiente .

El método PTF fue evaluado mediante un ambiente de simulación construido exprofeso. La simulación permitió observar el comportamiento del método PTF en diferentes circunstancias y comparar los planes producidos por él contra otros cuatro algoritmos de planificación, para los cuales también se construyeron ambientes de simulación. Sin embargo, estos ambientes de simulación no consideran el impacto producido por los costos de administración.

Del método PTF se observaron las siguientes ventajas y limitaciones:

Ventajas:

- PTF es una buena solución ante la carencia de técnicas que permitan planificar conjuntamente mezclas de tareas periódicas, aperiódicas y esporádicas flexibles.
- Como se comprobó mediante simulación, el método PTF es una estrategia de planificación de tareas flexibles que supera a los algoritmos de atención en trasfondo, atención en trasfondo mejorado, servidor diferido y servidor diferido mejorado cuando existen sobrecargas en el sistema.
- A diferencia de otros algoritmos, en los que una sobrecarga puede provocar una pérdida impredecible de los plazos de respuesta de las tareas, el método PTF es predecible en el sentido de que asegura el cumplimiento de los plazos de todas las tareas aceptadas en el sistema (las tareas periódicas y todas aquellas tareas aperiódicas y esporádicas que cumplen la MFC). Además, minimiza el error promedio de las tareas periódicas y el error total de las tareas aperiódicas con base en la capacidad de procesamiento asignada al SPAA.

Limitaciones:

- Aunque con PTF se asegura que ninguna tarea que cumpla la restricción de viabilidad de la parte obligatoria perderá su plazo de respuesta, es probable que en casos de una actividad aperiódica y esporádica intensa muchas tareas sean rechazadas. Sin embargo, el enfoque de atender estas tareas a través de una operación de manejo de excepciones (la cual debe ser simple para evitar impactar los costos de administración), es preferible al efecto impredecible ocasionado por la pérdida de un plazo de respuesta.
- El método PTF no es un algoritmo óptimo. Un algoritmo dinámico de planificación de tareas flexibles óptimo es aquel que, para cualquier conjunto de tareas flexibles que tenga planes viables, siempre encuentra un plan viable con error mínimo [Shih y Liu, 92]. Shih y Liu demostraron que no existen algoritmos óptimos en ese sentido, como se explicó en la sección 3.3.3.2. Por otro lado, en el capítulo 5 se comentó que el método PTF no es capaz de administrar la capacidad de procesamiento en trasfondo, por lo que algunas tareas que serían perfectamente planificables podrían ser rechazadas, haciendo imposible encontrar un plan viable con error mínimo en todos los casos.

En este trabajo también se estudiaron diversas estrategias para implantar computación flexible cuando el recurso restringido es el tiempo. Específicamente, algoritmos *anytime*, computación imprecisa y planificación *design-to-time*. Del análisis de estas estrategias se concluye que los algoritmos *anytime* y la computación imprecisa son ideas relacionadas y que la flexibilidad en ambos casos se encuentra en el diseño de las tareas. La planificación *design-to-time* es una alternativa a los enfoques anteriores, donde la flexibilidad en lugar de estar en las tareas se encuentra en los algoritmos de planificación.

Para aquellas aplicaciones que toleran el uso de resultados aproximados, por ejemplo, la transmisión y procesamiento progresivo de imagenes y video, el rastreo de objetivos a través de radares, y en particular los ejemplos de presentados en la sección 3.6 sobre sistemas de telecontrol de robots móviles, sistemas de control autónomo de vehículos, y la validación de sensores usando razonamiento probabilístico con algoritmos flexibles [Ibargüengoytia, 97], la computación flexible es una forma de disminuir el impacto de las sobrecargas, ya que, aunque los sistemas sufren una degradación paulatina, se evita la necesidad de realizar verificación frecuente de errores y ejecutar operaciones de recuperación de errores complejas.

La computación flexible es una alternativa para el desarrollo de los sistemas de tiempo real actuales, los cuales demandan importantes capacidades de procesamiento, ya que se puede aprovechar la capacidad de procesamiento disponible sin que se requiera un procesador más poderoso o un mayor número de procesadores para lograr su implantación.

Los sistemas de tiempo real son beneficiarios de las ideas y resultados producidos por la investigación sobre computación flexible, específicamente cuando el recurso restringido es el tiempo. Sin embargo, hacen falta trabajos sobre soporte de la computación flexible en diferentes sistemas operativos de tiempo real. En particular, en esta tesis sólo se tuvo acceso a [Hull y Liu, 93], [Hull et al., 95] y [Hull et al., 96], los cuales proponen soporte de computación flexible sobre el sistema operativo Mach.

También hacen falta trabajos que aborden la implantación de sistemas de tiempo real completos como conjuntos de tareas flexibles. Pocos fueron los trabajos encontrados en la literatura al respecto.

6.2 Trabajos futuros

A partir del trabajo realizado se vislumbra la posibilidad de llevar a cabo los siguientes trabajos adicionales:

- Desarrollar una estrategia con bajo costo de administración que permita administrar la capacidad de procesamiento disponible en trasfondo, con objeto de reducir el número de tareas aperiódicas y esporádicas que serían perfectamente planificables, pero que son rechazadas por el método PTF.
- Diseñar e implantar un simulador del método PTF en el que se considere el impacto producido por los costos de administración en la planificabilidad de las tareas.
- Hacer un análisis profundo sobre los mecanismos y estructuras de soporte a nivel sistema operativo necesarias para implantar el método PTF, y concluir este análisis con la implantación del método PTF en el *kernel* de un sistema operativo de tiempo real.
- Desarrollar métodos de planificación de tareas flexibles para modelos computacionales más complejos. Por ejemplo:
    - Haciendo una extensión al modelo aquí considerado en el que existan relaciones de precedencia entre las tareas, de modo que los resultados producidos por una tarea sean las entradas de otra tarea.
    - Una vez que se establece la existencia de relaciones de precedencia, considerar que el error de las tareas es una función del tiempo de ejecución asignado y del error existente en los datos de entrada.
- Extender los resultados obtenidos en este trabajo hacia la planificación de tareas flexibles en sistemas multiprocesadores.
- Desarrollar una simulación más profunda que permita determinar el impacto de cada una de las variables de entrada en el desempeño del método PTF.
- Estudiar la factibilidad de desarrollar trabajos que combinen proyectos sobre reconocimiento de voz y transmisión de imágenes de video, como los que se están desarrollando por [Uraga, 98] y [Canedo, 97], con el método de planificación desarrollado.

GLOSARIO

AER	Porcentaje de tareas aperiódicas y esporádicas rechazadas
EAE	Porcentaje de error de las tareas aperiódicas y esporádicas
EDF	Algoritmo de primero el plazo más próximo *(Earliest Deadline First)*
EP	Porcentaje de error de las tareas periódicas
LAT	Algoritmo de menor tiempo de atención *(Least Attained Time algorithm)*
MFC	Restricción de viabilidad de la parte obligatoria *(Mandatory Feasible Constraint)*
NORA	*No-Off-line tasks and on-line tasks Ready upon Arrival*
PTF	Método de planificación de tareas periódicas, aperiódicas y esporádicas flexibles
RMS	Algoritmo de razón monotónica *(Rate Monotonic Scheduling algorithm)*
SPAA	Servidor periódico de actividad aperiódica

costos de administración	*overhead*
planificador	*scheduler*
planificador removible	*preemptive scheduler*
plazo de respuesta	*deadline*
predecibilidad	*predictability*
tiempo de liberación	*ready time*

APÉNDICE A

DISEÑO DE PRUEBAS

Prueba 1

Conjunto de tareas periódicas.

Las partes obligatorias se encuentran resaltadas en negritas, no así las partes opcionales.

J_1: ***P(S_2), P(S_1), V(S_1), V(S_2), NOP***, *NOP, NOP, NOP*

J_2: ***P(S_3), P(S_2), NOP, V(S_2), V(S_3), NOP***, *NOP, NOP, NOP*

J_3: ***P(S_1), P(S_2), NOP, NOP, NOP, V(S_2), V(S_1), NOP***, *NOP, NOP*

Tabla A.1 Conjunto de tareas periódicas de la prueba 1.

Tarea	***Período (π_i)***	***Tiempo de ejecución (τ_i)***	***Parte obligatoria (m_i)***	***Parte opcional (o_i)***	***Tiempo máximo de bloqueo (B_i)***
J_1	40	8	5	3	7
J_2	50	9	6	3	5
J_3	60	10	8	2	0

Servidor periódico de actividad aperiódica.

$\tau_o = 7;\ \pi_o = 39$

Características de las tareas aperiódicas.

$1/\lambda = 100;\ 1/\mu_m = 2;\ 1/\mu_o = 1$

Características de las tareas esporádicas.

Tabla A.2 Conjunto de tareas esporádicas de la prueba 1.

Tarea	***Tiempo de ejecución (C_i)***	***Parte obligatoria (ob_i)***	***Parte opcional (op_i)***	***Mínimo tiempo entre arribos (mta_i)***
E_1	3	1	2	200
E_2	3	2	1	90

Longitud de simulación.

20007 unidades de tiempo (en ésta y las siguientes pruebas, la longitud de simulación es el primer múltiplo de π_o mayor o igual a 20000).

Prueba 2

Conjunto de tareas periódicas y servidor periódico de actividad aperiódica.

Los mismos que para la prueba 1.

Características de las tareas aperiódicas.

$1/\lambda = 100;\ 1/\mu_m = 2;\ 1/\mu_o = 8$

Características de las tareas esporádicas.

Tabla A.3 Conjunto de tareas esporádicas de la prueba 2.

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	5	1	4	200
E_2	8	2	6	90

Longitud de simulación.

20010 unidades de tiempo.

Prueba 3

Conjunto de tareas periódicas y servidor periódico de actividad aperiódica.

Los mismos que para la prueba 1.

Características de las tareas aperiódicas.

$1/\lambda = 30;\ 1/\mu_m = 3;\ 1/\mu_o = 3$

Características de las tareas esporádicas.

Tabla A.4 Conjunto de tareas esporádicas de la prueba 3.

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	4	2	2	50
E_2	6	4	2	60

Longitud de simulación.

20007 unidades de tiempo.

Prueba 4

Conjunto de tareas periódicas.

El mismo que para la prueba 1.

Servidor periódico de actividad aperiódica.

$\tau_o' = 14;\ \pi_o = 39$

Características de las tareas aperiódicas y esporádicas.

Las mismas que para la prueba 3.

Longitud de simulación.

20007 unidades de tiempo.

Prueba 5

Conjunto de tareas periódicas.

J_1: ***NOP, P(S_1), NOP, V(S_1), NOP***

J_2: ***P(S_1), NOP, V(S_1), P(S_2), V(S_2), NOP***

J_3: ***P(S_3), NOP, P(S_2), NOP, V(S_2), V(S_3), NOP***

Tabla A.5 Conjunto de tareas periódicas de la prueba 5.

Tarea	*Período (π_i)*	*Tiempo de ejecución (τ_i)*	*Parte obligatoria (m_i)*	*Parte opcional (o_i)*	*Tiempo máximo de bloqueo (B_i)*
J_1	30	5	5	0	3
J_2	35	6	6	0	3
J_3	40	7	7	0	0

Servidor periódico de actividad aperiódica.

$\tau_o = 6;\ \pi_o = 29$

Características de las tareas aperiódicas.

$1/\lambda = 100;\ 1/\mu_m = 1;\ 1/\mu_o = 2$

Características de las tareas esporádicas.

Tabla A.6 Conjunto de tareas esporádicas de la prueba 5.

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	5	2	3	150

Longitud de simulación.

20010 unidades de tiempo.

Prueba 6

Conjunto de tareas periódicas y servidor periódico de actividad aperiódica.

Los mismos que para la prueba 5.

Características de las tareas aperiódicas.

$1/\lambda = 20;\ 1/\mu_m = 1;\ 1/\mu_o = 3$

Características de las tareas esporádicas.

Tabla A.7 Conjunto de tareas esporádicas de la prueba 6.

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	5	2	3	50

Longitud de simulación.

20010 unidades de tiempo.

Prueba 7

Conjunto de tareas periódicas y servidor periódico de actividad aperiódica.

Los mismos que para la prueba 5.

Características de las tareas aperiódicas.

$1/\lambda = 20;\ 1/\mu_m = 4;\ 1/\mu_o = 4$

Características de las tareas esporádicas.

Tabla A.8 Conjunto de tareas esporádicas de la prueba 7.

Tarea	***Tiempo de ejecución (C_i)***	***Parte obligatoria (ob_i)***	***Parte opcional (op_i)***	***Mínimo tiempo entre arribos (mta_i)***
E_1	7	4	3	35

Longitud de simulación.

20010 unidades de tiempo.

Prueba 8

Conjunto de tareas periódicas.

J_1: ***P(S_1), V(S_1), NOP,*** *NOP, NOP, NOP, NOP*

J_2: ***P(S_2), V(S_2), P(S_1), V(S_1), NOP,*** *NOP, NOP, NOP, NOP, NOP*

J_3: ***P(S_2), V(S_2), NOP,*** *NOP, NOP, NOP, NOP, NOP*

Tabla A.9 Conjunto de tareas periódicas de la prueba 8.

Tarea	***Período (π_i)***	***Tiempo de ejecución (τ_i)***	***Parte obligatoria (m_i)***	***Parte opcional (o_i)***	***Tiempo máximo de bloqueo (B_i)***
J_1	35	7	3	4	2
J_2	40	10	5	5	2
J_3	41	8	3	5	0

Servidor periódico de actividad aperiódica.

$\tau_o' = 12;\ \pi_o = 34$

Características de las tareas aperiódicas.

$1/\lambda = 50;\ 1/\mu_m = 2;\ 1/\mu_o = 3$

Características de las tareas esporádicas.

Tabla A.10 Conjunto de tareas esporádicas de la prueba 8.

Tarea	***Tiempo de ejecución (C_i)***	***Parte obligatoria (ob_i)***	***Parte opcional (op_i)***	***Mínimo tiempo entre arribos (mta_i)***
E_1	5	3	2	60
E_2	4	1	3	100

Longitud de simulación.

20026 unidades de tiempo.

Prueba 9

Conjunto de tareas periódicas.

J_1: ***P(S_1), P(S_2), NOP, NOP, V(S_2), V(S_1), NOP,*** *NOP, NOP, NOP, NOP, NOP*

J_2: ***P(S_2), P(S_1), NOP, V(S_1),V(S_2), NOP,*** *NOP, NOP, NOP, NOP, NOP*

J_3: ***P(S_1), NOP, NOP, NOP, NOP, NOP, V(S_1), NOP,*** *NOP, NOP, NOP, NOP, NOP, NOP*

Tabla A.11 Conjunto de tareas periódicas de la prueba 9.

Tarea	*Período (π_i)*	*Tiempo de ejecución (τ_i)*	*Parte obligatoria (m_i)*	*Parte opcional (o_i)*	*Tiempo máximo de bloqueo (B_i)*
J_1	35	12	7	5	7
J_2	40	11	6	5	7
J_3	45	14	8	6	0

Servidor periódico de actividad aperiódica.

$\tau_o' = 6;\ \pi_o = 34$

Características de las tareas aperiódicas.

$1/\lambda = 120;\ 1/\mu_m = 1;\ 1/\mu_o = 1$

Características de las tareas esporádicas.

Tabla A.12 Conjunto de tareas esporádicas de la prueba 9.

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	4	2	2	180

Longitud de simulación.

20026 unidades de tiempo.

Prueba 10x

Conjunto de tareas periódicas.

J_1: ***P(S_2), P(S_1), V(S_1), V(S_2), NOP,*** *NOP, NOP, NOP, NOP, NOP, NOP, NOP, NOP, NOP, NOP, NOP*

J_2: ***P(S_3), P(S_2), NOP, V(S_2),V(S_3), NOP,*** *NOP, NOP, NOP, NOP, NOP, NOP, NOP, NOP, NOP*

J_3: ***P(S_1), P(S_2), NOP, NOP, NOP, V(S_2), V(S_1), NOP,*** *NOP, NOP, NOP, NOP*

Tabla A.13 Conjunto de tareas periódicas de la prueba 10x.

Tarea	*Período (π_i)*	*Tiempo de ejecución (τ_i)*	*Parte obligatoria (m_i)*	*Parte opcional (o_i)*	*Tiempo máximo de bloqueo (B_i)*
J_1	40	16	5	11	7
J_2	50	15	6	9	7
J_3	60	12	8	4	0

Servidor periódico de actividad aperiódica.

$\pi_o = 39.$

Tabla A.14 Capacidades del SPAA.

	Capacidad del SPAA
Prueba 10a	$\tau_o' = 14$
Prueba 10b	12
Prueba 10c	10
Prueba 10d	8
Prueba 10e	6

Características de las tareas aperiódicas.

$1/\lambda = 50;\ 1/\mu_m = 3;\ 1/\mu_o = 11$

Características de las tareas esporádicas.

Tabla A.15 Conjunto de tareas esporádicas de la prueba 10x.

Tarea	***Tiempo de ejecución (C_i)***	***Parte obligatoria (ob_i)***	***Parte opcional (op_i)***	***Mínimo tiempo entre arribos (mta_i)***
E_1	9	2	7	60
E_2	13	3	10	150

Longitud de simulación.

20007 unidades de tiempo.

Prueba 11*x*

Conjunto de tareas periódicas.

J_1: ***NOP, P(S_1), NOP, V(S_1), NOP,*** *NOP*

J_2: ***P(S_1), NOP, V(S_1), P(S_2), V(S_2), NOP,*** *NOP, NOP*

J_3: ***P(S_3), NOP, P(S_2), NOP, V(S_2), V(S_3), NOP,*** *NOP, NOP, NOP*

Tabla A.16 Conjunto de tareas periódicas de la prueba 11x.

Tarea	***Período (π_i)***	***Tiempo de ejecución (τ_i)***	***Parte obligatoria (m_i)***	***Parte opcional (o_i)***	***Tiempo máximo de bloqueo (B_i)***
J_1	30	5	5	1	3
J_2	35	6	6	2	3
J_3	40	7	7	3	0

Servidor periódico de actividad aperiódica.

$\pi_o = 29.$

Tabla A.17 Capacidades del SPAA

	Capacidad del SPAA
Prueba 11a	$\tau_o' = 6$
Prueba 11b	5
Prueba 11c	4
Prueba 11d	3

Características de las tareas aperiódicas.

$1/\lambda = 100;\ 1/\mu_m = 2;\ 1/\mu_o = 5$

Características de las tareas esporádicas.

Tabla A.18 Conjunto de tareas esporádicas de la prueba 11x.

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	6	1	5	200
E_2	8	2	6	90

Longitud de simulación.

20010 unidades de tiempo.

Prueba 12

Conjunto de tareas periódicas.

*J_1: **NOP, NOP, P(S_1), NOP, V(S_1), NOP**, NOP, NOP*

*J_2: **NOP, P(S_1), NOP, V(S_1), NOP, NOP, NOP**, NOP, NOP, NOP*

*J_3: **NOP, P(S_1), P(S_2), V(S_2), V(S_1), NOP**, NOP, NOP, NOP*

*J_4: **NOP, NOP, NOP**, NOP, NOP, NOP*

Tabla A.19 Conjunto de tareas periódicas de la prueba 12.

Tarea	*Período (π_i)*	*Tiempo de ejecución (τ_i)*	*Parte obligatoria (m_i)*	*Parte opcional (o_i)*	*Tiempo máximo de bloqueo (B_i)*
J_1	40	8	6	2	4
J_2	45	10	7	3	4
J_3	50	9	6	3	0
J_4	55	6	3	3	0

Servidor periódico de actividad aperiódica.

$\tau_o' = 9;\ \pi_o = 39$

Características de las tareas aperiódicas.

$1/\lambda = 30;\ 1/\mu_m = 4;\ 1/\mu_o = 2$

Características de las tareas esporádicas.

Tabla A.20 Conjunto de tareas esporádicas de la prueba 12.

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	6	4	2	60
E_2	7	3	4	90

Longitud de simulación.

20007 unidades de tiempo.

Prueba 13

Conjunto de tareas periódicas.

*J_1: **NOP, NOP, NOP, NOP**, NOP, NOP, NOP*

*J_2: **NOP, NOP, NOP, NOP, NOP**, NOP, NOP, NOP, NOP*

*J3: **NOP, NOP, NOP, NOP, NOP, NOP, NOP**, NOP, NOP, NOP*

Tabla A.21 Conjunto de tareas periódicas de la prueba 13.

Tarea	*Período* (π_i)	*Tiempo de ejecución* (τ_i)	*Parte obligatoria* (m_i)	*Parte opcional* (o_i)	*Tiempo máximo de bloqueo* (B_i)
J_1	25	7	4	3	0
J_2	35	9	5	4	0
J_3	40	10	7	3	0

Servidor periódico de actividad aperiódica.

$\tau_o' = 6;\ \pi_o = 24$

Características de las tareas aperiódicas.

$1/\lambda = 35;\ 1/\mu_m = 5;\ 1/\mu_o = 3$

Características de las tareas esporádicas.

Tabla A.22 Conjunto de tareas esporádicas de la prueba 13.

Tarea	*Tiempo de ejecución* (C_i)	*Parte obligatoria* (ob_i)	*Parte opcional* (op_i)	*Mínimo tiempo entre arribos* (mta_i)
E_1	3	2	1	50
E_2	4	2	2	80

Longitud de simulación.

20016 unidades de tiempo.

Prueba 14

Conjunto de tareas periódicas.

J_1: ***NOP, P(S_1), NOP, V(S_1), NOP,*** *NOP, NOP*

J_2: ***NOP, NOP, P(S_1), V(S_1), NOP,*** *NOP, NOP,NOP*

Tabla A.23 Conjunto de tareas periódicas de la prueba 14.

Tarea	*Período* (π_i)	*Tiempo de ejecución* (τ_i)	*Parte obligatoria* (m_i)	*Parte opcional* (o_i)	*Tiempo máximo de bloqueo* (B_i)
J_1	50	7	5	2	2
J_2	55	8	5	3	0

Servidor periódico de actividad aperiódica.

$\tau_o' = 20;\ \pi_o = 49$

Servidor diferido.

$\tau_o = 17;\ \pi_o = 49$

Características de las tareas aperiódicas.

$1/\lambda = 45;\ 1/\mu_m = 2;\ 1/\mu_o = 3$

Características de las tareas esporádicas.

Tabla A.24 Conjunto de tareas esporádicas de la prueba 14.

Tarea	*Tiempo de ejecución* (C_i)	*Parte obligatoria* (ob_i)	*Parte opcional* (op_i)	*Mínimo tiempo entre arribos* (mta_i)
E_1	6	3	3	40

Longitud de simulación.

20041 unidades de tiempo.

Prueba 15

Conjunto de tareas periódicas, servidor periódico de actividad aperiódica y servidor diferido.

Los mismos que para la prueba 14.

Características de las tareas aperiódicas.

$1/\lambda = 45$; $1/\mu_m = 2$; $1/\mu_o = 10$

Características de las tareas esporádicas.

Tabla A.25 Conjunto de tareas esporádicas de la prueba 15.

Tarea	*Tiempo de ejecución (C_i)*	*Parte obligatoria (ob_i)*	*Parte opcional (op_i)*	*Mínimo tiempo entre arribos (mta_i)*
E_1	9	3	6	40

Longitud de simulación.

20041 unidades de tiempo.

Prueba 16

Similar a la prueba 1, pero con $\tau_o' = 14$.

Servidor diferido.

$\tau_o = 7$; $\pi_o = 39$

Prueba 17

Similar a la prueba 2, pero con $\tau_o' = 14$.

Servidor diferido.

$\tau_o = 7$; $\pi_o = 39$

Prueba 18

Idéntica a la prueba 9.

Servidor diferido.

$\tau_o = 0$.

Prueba 19

Idéntica a la prueba 10a.

Servidor diferido.

$\tau_o = 0$.

REFERENCIAS BIBLIOGRÁFICAS

[Baruah y Haritsa, 97] S. K. Baruah y J. R. Haritsa. Scheduling for overload in real-time systems. *IEEE Transactions on Computers*, Vol. 4, No. 9. September, 1997.

[Burns, 91] A. Burns. Scheduling hard real-time systems: a review. *Software Engineering Journal*, Vol. 6, No. 3. May, 1991.

[Burns y Wellings, 90] A. Burns y A. Wellings. Real-time systems and their programming languages. Addison-Wesley Publishing Company. UK, 1990.

[Canedo, 97] E. Canedo. Integración de video en una red de cómputo. Reporte de avance de Tesis de Maestría en Ciencias Computacionales. ITESM, Campus Morelos. Diciembre, 1997.

[Chung y Liu, 88] J.-Y. Chung y J. W. S. Liu. Algorithms for scheduling periodic jobs to minimize average error. 1988.

[Chung y Liu, 89] J.-Y. Chung y J. W. S. Liu. Performance of algorithms for scheduling periodic jobs to avoid timing faults. 1989.

[Chung et al., 90] J.-Y. Chung, J. W. S. Liu y K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, Vol. 39, No. 9, September, 1990.

[Cozman, 96] F. Cozman. Control of computational effort trough robustness analysis: a study of visual position estimation under time-critical constraints. *AAAI-96 Fall Symposium on Flexible Computation in Intelligent Systems: Results, Issues and Oportunities.* Cambridge, Massachusetts. November, 1996.

[Davis et al., 95] R. Davis, S. Punnekkat, N. Audsley y A. Burns. Flexible scheduling for adaptable real-time systems. *Proceedings of the IEEE Real-Time Technology and Applications Symposium.* May, 1995.

[Dean y Boddy, 88] T. Dean y M Boddy. An analysis of time-dependent planning. *Proceedings of the Seventh National Conference on Artificial Intelligence*. Minneapolis, Minnesota, 1988.

[Feng y Liu, 97] W. Feng y J. W. S. Liu. Algorithms for scheduling real-time tasks with input error and end-to-end deadlines. *IEEE Transactions on Software Engineering*. Vol. 23, No. 2. February, 1997.

[Garvey y Lesser, 96] A. Garvey y V. Lesser. Issues in design-to-time real-time scheduling. *AAAI-96 Fall Symposium on Flexible Computation in Intelligent Systems: Results, Issues and Oportunities*. Cambridge, Massachusetts. November, 1996.

[Grass, 96] J. Grass. Reasoning about computational resource allocation, an introduction to anytime algorithms. *ACM Crossroads*. September, 1996.

[Gross y Harris, 85] D. Gross y C. M. Harris. Fundamentals of queueing theory. John Wiley & Sons. Second edition, 1985.

[Halang y Sacha, 92] W. A. Halang y K. M. Sacha. Real-time systems: implementation of industrial computerised process automation. World Scientific Publishing Co. Singapur, 1992.

[Horowitz y Sahni, 78] E. Horowitz y S. Sahni. Fundamentals of computer algorithms. Computer Science Press. USA, 1978.

[Horvitz, 87] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*. Seattle, Wa. July, 1987.

[Horvitz, 97] E. J. Horvitz. Flexible Computation [Online]. Available: *http://www.research.microsoft.com/research/dtg/horvitz/flex.htm*. [June, 1997].

[Hull y Liu, 93] D. Hull y J. W. S. Liu . ICS: a system for imprecise computations. American Institute of Aeronautics, Inc. 1993.

[Hull et al., 95].D. L. Hull, W. Feng y J. W .S. Liu. Enhancing the performance and dependability of real-time systems. *IEEE International Computer Performance and Dependability.* Erlangen, Germany. April, 1995.

[Hull et al., 96] D. L. Hull, W. Feng y J. W .S. Liu. Operating system support for imprecise computation. *AAAI-96 Fall Symposium on Flexible Computation in Intelligent Systems: Results, Issues and Oportunities.* Cambridge, Massachusetts. November, 1996.

[Ibargüengoytia, 97] P. H. Ibargüengoytia. Any time probabilistic sensor validation. Ph.D. dissertation. Department of Computer & Mathematical Sciences. TIME Research Institute. University of Salford, UK. October, 1997.

[Klein et al., 93] M. H. Klein, T. Ralya, B. Pollak, R. Obenza y M. González. A practitioners handbook for real-time analysis: guide to rate monotonic analysis for real-time systems. Kluwer Academic Publishers. 1993.

[Lehoczky et al., 89] J. Lehoczky, L. Sha y Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. *Proceedings of the 10th IEEE Real-Time Systems Symposium*, 1989.

[Lehoczky et al., 91] J. P. Lehoczky, L. Sha, J. K. Strosnider y H. Tokuda. Fixed priority scheduling theory for hard-real time systems. *Foundations of Real-Time Computing: Scheduling and Resource Management.* A. M. van Tilborg y G. M. Koob editors. Kluwer Academic. New York, 1991

[Lin y Tarng, 91] T.-H. Lin y W. Tarng. Scheduling periodic and aperiodic tasks in hard real-time computing systems. ACM, 1991.

[Liu et al., 91] J. W. S. Liu, K.-J. Lin, W.-K. Shih y A. C. Yu. Algorithms for scheduling imprecise computations. *IEEE Computer.* May, 1991.

[Liu et al., 94] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati y J.-Y. Chung. Imprecise Computations. *Proceedings of the IEEE*, Vol. 82, No. 1, January, 1994.

[Liu y Layland, 73] C. L. Liu y J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, Vol 20, No. 1. January, 1973.

[Meyer, 86] P. L. Meyer. Probabilidad y aplicaciones estadísticas. Addison Wesley Iberoamericana. México, 1986.

[Milenković, 94] M. Milenković. Sistemas operativos, conceptos y diseño. McGraw-Hill. Second edition (spanish). Madrid, 1994.

[Myers, 76] G. J. Myers. Software reliability, principles and practices. John Wiley & Sons. 1976.

[Ramamritham y Stankovic, 94] K. Ramamrithan y J. A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, Vol. 82, No. 1. January, 1994.

[Russell y Zilberstein, 91] S. J. Russell y S. Zilberstein. Composing real-time systems. *Proceedings of the 12th International Joint Conference on Artificial Intelligence.* Sydney, Australia. August, 1991.

[Sha et al., 90] L. Sha, R. Rajkumar y J. P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *Transactions on Computers*, Vol. 39, No. 9. September, 1990.

[Sha et al., 94] L. Sha, R. Rajkumar y S. S. Sathaye. Generalized rate-monotonic scheduling theory: a framework for developing real-time systems. *Proceedings of the IEEE*, Vol. 82., No. 1. January, 1994.

[Shapiro, 92] S. Shapiro. Encyclopedia of artificial intelligence, volume 1. Second edition. John Wiley & Sons, Inc. USA, 1992.

[Shih y Liu, 92] W.-K. Shih y J. W. S. Liu. On-line scheduling of imprecise computations to minimize error. *Proceedings of the 13th IEEE Real-Time Systems Symposium*. Phoenix, AZ. December, 1992.

[Shih y Liu, 95] W.-K. Shih y J. W. S. Liu. Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error. *IEEE Transactions on Computers*, Vol. 44, No. 3. March, 1995.

[Shin y Ramanathan, 94] K. G. Shin y P. Ramanathan. Real-time computing: a new discipline of computer Science and engineering. *Proceedings of the IEEE*, Vol. 82. No. 1. January, 1994.

[Singhal y Shivaratri, 94] M. Singhal y N. G. Shivaratri. Advanced concepts in operating systems. McGraw-Hill. USA, 1994.

[Stallings, 97] W. Stallings. Sistemas Operativos. Second edition (spanish). Prentice Hall. Madrid, 1997.

[Stankovic et al., 95] J. A. Stankovic, M. Spuri, M. Di Natale y G. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, Vol. 28, No. 6. June, 1995.

[Strosnider et al., 95] J. K. Strosnider, J. P. Lehoczky y L. Sha. The deferreable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on computers*. Vol. 44, No. 1. January, 1995.

[Tamer y Valduriez, 91] M. Tamer y P. Valduriez. Principles of distributed database systems. Prentice-Hall. USA, 1991.

[Uraga, 98] E. Uraga. Desarrollo de un sistema de reconocimiento automático de voz. Tesis de Maestría en Ciencias Computacionales. ITESM, Campus Morelos, 1998 (en desarrollo).