Análisis de Sistemas de Tiempo Real Duro con Constricciones de Precedencia.

Ricardo Luis Cayssials

Tesis presentada en cumplimiento
parcial de los requerimientos para el
Doctorado en Ingeniería

Departamento de Graduados
Departamento de Ingeniería Eléctrica
Universidad Nacional del Sur

Agradecimientos

Soy consciente que, sin el apoyo que he recibido, hubiera sido incapaz de comenzar mi tesis y mucho menos terminarla. Sé también, y por eso pido disculpas, que mi memoria me va a fallar en este momento, para recordar todas y cada una de las personas que me ayudaron a lograrlo. Sin embargo, aunque omita algún nombre en forma involuntaria, me siento tranquilo porque el afecto de las personas a quien quiero es mucho mayor que un torpe descuido mío.

Deseo agradecer especialmente al Ingeniero Jorge Santos quien creyó en mí y cuya paciencia, ejemplo y afecto me estimulan continuamente a mejorar profesionalmente y personalmente. Le agradezco la tolerancia que demostró ante las preocupaciones que le generé y por todo el apoyo, siempre basado en la confianza, que me brindó en todo momento.

A mis actuales compañeros de trabajo: Edgardo Ferro, Javier Orozco, Omar Alimenti y Rodrigo Santos por haber compartido discusiones e ideas, pero especialmente por disimular todas mis manías en el mucho tiempo que compartimos diariamente.

A Edgardo Ferro por ser el "culpable" de que haya elegido la actividad académica.

A Javier Orozco por tolerar todas mis preguntas y por estar siempre dispuesto a ayudarme.

Al Profesor Julio Sewald y a la Licenciada Graciela Paolini.

A los Profesores Gallard y Baum, jurados de esta tesis, cuyas observaciones contribuyeron a mejorarla.

A la Familia Di Cicco de la que recibí mucho cariño. Gracias a Gabriel y Perla por tratarme como a un hijo y por estimular, con su afecto, mi corazón.

A Germana, Luisa y Enrique por haberme malcriado como sólo los abuelos pueden hacerlo.

A Nené, Juan, Juan Pablo y Moni por estar siempre cerca.

Deseo dedicar esta tesis a mi familia, especialmente a mis padres por haber luchado y trabajado tanto para brindarle a sus hijos una vida mejor de la que ellos mismos tuvieron. Quiero agradecerles su confianza y amor incondicional y por sentirse siempre orgullosos de las pequeñas cosas que logra su hijo.

A Marcelo, por ser mi hermano mayor, amigo, por su contención y comprensión y por sentirlo siempre cerca, aún en la distancia.

A Silvita por alegrar mi vida, por su cariño y confianza y porque me deja ser su hermano.

A todos ellos y a quienes involuntariamente omití, sinceramente gracias.

Prefacio

Esta tesis ha sido desarrollada en el Laboratorio de Sistemas Digitales del Departamento de Ingeniería Eléctrica de la Universidad Nacional del Sur. Según se ha adoptado como norma para los estudios de posgrado en dicho Laboratorio, esta tesis está basada en publicaciones de las cuales soy autor principal o coautor. Estas publicaciones y/o el contenido de esta tesis no han sido ni serán utilizados como contenido de otra tesis llevada a cabo en dicho ámbito. Las principales publicaciones en orden cronológico inverso son: **Cayssials**, R., Orozco, J., Santos, J., and Santos, R., "Rate Monotonic scheduling of real-time control systems with the minimum number of priority levels", *Proc. 11th Euromicro Workshop on Real Time Systems '99*, IEEE Computer Society Press, York, England, 1999, (en prensa); **Cayssials**, R., Santos, J., Orozco, J., and Ferro, E., "On the Scheduling of Real-Time Multihop Packet-Switching Networks", *Proc. 10th Euromicro Workshop on Real Time Systems '98*, IEEE Computer Society Press, Berlin, 1998; Orozco, J., **Cayssials**, R., Santos, J., and Ferro, E., "Precedence Constraints in Hard Real Time Distributed Systems", *Proc. Third IEEE International Conference on Engineering of Complex Computer Systems*, IEEE Computer Society Press, Italy, 1997; Orozco, J., **Cayssials**, R., Santos, J., and Ferro, E., "Design of a Learning Fuzzy Production System to Solve An NP-Hard Real-Time Assignment Problem", *Proc.8th Euromicro Workshop on Real Time Systems '96*, IEEE Computer Society Press, Italy, 1996.

Ricardo Cayssials
Bahía Blanca, abril de 1999

Indice

Introducción

En el mundo actual estamos en continuo contacto con sistemas que necesitan procesar datos a intervalos regulares de tiempo para su correcto funcionamiento. Por ejemplo, un avión usa una secuencia de pulsos de un acelerómetro para determinar su posición. Además otros sistemas requieren una "rápida" respuesta a eventos que ocurren a intervalos no regulares, como puede ser la respuesta a una falla de sobretemperatura en un reactor atómico. Estos sistemas deben responder en forma correcta, segura y confiable antes de un determinado tiempo. En sistemas de *tiempo real duro* la respuesta, aunque sea correcta desde el punto de vista lógico aritmético, puede producir resultados imprevisibles si se produce tardíamente. Las consecuencias por no cumplir con las restricciones temporales, generalmente son ilustradas en los trabajos de investigación o en la bibliografía de tiempo real duro como causante de pérdidas de vidas humanas, destrucción total de aviones, reactores nucleares o catástrofes de gravedad similar. Sin embargo, existen sistemas en que la respuesta tardía produce sólo un malfuncionamiento pero no causa ninguna secuela de las magnitudes antes mencionadas. Para el diseñador de estos últimos sistemas,

las características de tiempo real duro cobran la misma importancia que en los ejemplos caóticos tradicionales, si se desea obtener un sistema que funcione correctamente.

Una parte esencial en los sistemas de tiempo real es la que cumple la función de diagramar las diferentes tareas que lo integran. El principal problema es que las distintas funciones que debe cumplir el sistema son llevadas a cabo por sendas tareas que el diseñador programa *ad hoc* y deben ejecutarse generalmente en un número menor de procesadores. Para que todas las tareas puedan desarrollar sus funciones, se le otorga el procesador a cada una de ellas durante un intervalo predeterminado de tiempo, siendo la función del diagramador administrar su uso estableciendo la tarea que lo utilizará en cada uno de los intervalos. Una mala administración del procesador puede causar la respuesta tardía de una o varias tareas, provocando un mal funcionamiento de todo el sistema.

Generalmente el diagramador es implementado por una tarea del Sistema Operativo cuya función es ejecutar una disciplina de prioridades para determinar, del conjunto de tareas que estén requiriendo ser ejecutadas en el procesador, la tarea a la que efectivamente se le adjudicará por un intervalo de tiempo predefinido. Según sea la disciplina de prioridades o la determinación del intervalo que cada tarea tendrá a su disposición al procesador, pueden establecerse diferentes tipos de diagramadores de mayor o menor eficacia, confiabilidad o simplicidad. Si el sistema consta de varios procesadores en donde pueden ejecutarse las diferentes tareas, entonces existen dos alternativas para la implementación del diagramador: la primera es un único diagramador que controla la ejecución de todas las tareas en todos los procesadores, mientras que la segunda es un diagramador por procesador que administra un subconjunto de tareas que fueron asignadas a dicho procesador en la etapa de diseño. En la primera alternativa se define que el diagramador es *global*, opuestamente a la segunda en donde se dice que el diagramador es *local*.

Las características propias de cada uno de estos tipos de diagramadores, los hacen aptos para ciertas aplicaciones e ineficientes o inseguros en otras.

Por organización, confiabilidad o necesidad, el diseñador puede optar por implementar cada función del sistema por un conjunto de tareas cooperativas entre ellas. De esta manera, dicho conjunto de tareas cooperativas que persiguen un objetivo en común, definen el concepto de *trabajo*. Una tarea perteneciente a un trabajo puede necesitar, para cumplir su función, la ejecución previa de otra tarea que pertenece al mismo trabajo. Esta restricción que imposibilita la ejecución de una tarea sin la ejecución previa de otra, determina relaciones de precedencia entre ellas. Las relaciones de precedencia insertan particularidades tan significativas, tanto en sistemas monoprocesador o multiprocesador, que pueden determinar la factibilidad, o no, de todo el sistema en tiempo real duro.

Todos los factores antes mencionados (disciplina de prioridades, cantidad de procesadores, elección del diagramador local o global, precedencias entre las tareas) brindan un espectro muy amplio para la implementación de sistemas con diferentes características en su confiabilidad, eficiencia, seguridad o flexibilidad para adaptarse a futuras modificaciones. Esta tesis, tratará en sus resultados inéditos principalmente, el estudio de la disciplina de Prioridades Fijas en diagramadores locales (cuando se trate de sistemas multiprocesador) con tareas con relaciones de precedencia. La elección de cada una de estas características se explicará con mayor detalle cuando su influencia en la investigación lo justifique.

Esta tesis está organizada de la siguiente manera: en el capítulo 1 se describen los términos y definiciones utilizadas en tiempo real. En el capítulo 2 se desarrollan los principales conceptos de la diagramación de sistemas, como distintas alternativas de implementación. El capítulo 3 introduce el concepto de pesimismo en el análisis de la diagramabilidad y enuncia las diferentes condiciones obtenidas para diagramadores en tiempo

real. El capítulo 4 abarca los principales resultados obtenidos en investigaciones sobre sistemas monoprocesador y multiprocesador, diagramados por Prioridades Fijas, principalmente el método de las ranuras vacías. En el capítulo 5 se demuestra una condición suficiente para analizar la diagramabilidad de un sistema monoprocesador con tareas con relaciones de precedencia y se muestra la conveniencia de este tipo de relaciones en la diagramación de sistemas con Prioridades Fijas. En el capítulo 6 se pasa a los sistemas multitarea-multiprocesador. En el mismo se describe un método no-convencional de asignación con base en lógicas difusas, se obtiene una condición de diagramabilidad para validar una asignación tentativa cuando las tareas tienen condiciones de precedencia y, finalmente, se propone un mecanismo para temporizar la generación de las tareas que constituyen un trabajo. En el capítulo 7 se plantea un isomorfismo entre sistemas multiprocesador y redes de conmutación de paquetes, para utilizar las condiciones de tiempo real del capítulo anterior en este tipo de redes. En el capítulo 8 se exponen las conclusiones y se proponen los futuros trabajos de investigación que deberían realizarse basados en los obtenidos en esta tesis. En el apéndice A se resumen todos los símbolos y definiciones utilizados. El apéndice B contiene el enunciado de todos los lemas y teoremas utilizados y desarrollados. La última sección enumera las referencias utilizadas.

Capítulo 1

Este capítulo es una introducción a los principales conceptos de tiempo real. Se presentan los términos y las definiciones utilizadas en el área de tiempo real, poniendo énfasis en la diagramación de sistemas multitarea.

1. Conceptos de Tiempo Real

Un sistema de tiempo real es aquél en el cual los aspectos de comportamiento temporal forman parte de su especificación. El correcto funcionamiento de estos sistemas no sólo depende de la exactitud de los resultados aritméticos y lógicos, sino también del momento en que estos resultados son producidos. Esto ocurre frecuentemente en sistemas que deben interactuar con un entorno que se modifica constantemente, como es el caso de sistemas de control de procesos, robótica, telecomunicaciones, multimedia y realidad virtual, entre otros.

Es característica principal de los programas de estos sistemas, la permanente supervisión de los datos de entrada, para reaccionar en consecuencia cuando correspondiere. Esta reacción debe finalizar antes de un determinado tiempo, denominado *vencimiento* (*deadline* en inglés). En

esta tesis se utilizará indiferentemente la expresión *perder un vencimiento* o *entrar en crisis* para expresar que el vencimiento no se ha cumplido. Las consecuencias que pueden provocar el no cumplimiento de algunos vencimientos divide a los sistemas de tiempo real en *duros* y *blandos*. En los sistemas de tiempo real duro es imperativo que ningún vencimiento sea excedido, mientras que en los denominados blandos el sistema tolera el no cumplimiento esporádico de algunos vencimientos. Como un buen ejemplo de sistemas de tiempo real blando puede citarse la transmisión de multimedia, en la cual la pérdida ocasional de algún cuadro de video no trae aparejada grandes consecuencias. Por otro lado, un ejemplo de tiempo real duro es un simple sistema de control, en el que una respuesta tardía del sistema provoca una falla o ineficiencia en el control del proceso. Generalmente los ejemplos tradicionales de la bibliografía de tiempo real duro, ilustran estas fallas con la destrucción total del sistema y pérdidas de vidas humanas, con la intención de dar énfasis a la importancia que tiene el cumplimiento de todos los vencimientos. Dichos ejemplos, si bien son efectivos al momento de ilustrar la importancia que poseen los vencimientos del sistema, generan inconscientemente la creencia que los sistemas de tiempo real duro deben ser utilizados solamente en ambientes críticos. Sin embargo, existen sistemas en los que las condiciones temporales son de tiempo real duro, pero las fallas del sistema sólo provocan una degradación en su eficiencia o lo colocan en situaciones indeseadas. Como ejemplo de esto, se puede citar un sistema para pronosticar el estado del clima con un día de anticipación. Dicho sistema no será útil, por no cumplir las condiciones de tiempo real duro, si demora más de 24 horas en entregar el resultado.

Es coherente pensar que, en un sistema de tiempo real duro, no todas las reacciones a los estímulos externos provocan el mismo grado de crisis ante una respuesta tardía. De esta manera, algunas respuestas tardías provocarán un mal funcionamiento y eventualmente una crisis severa del

sistema, mientras que otras provocarán sólo una degradación de su eficiencia.

El desarrollo de un sistema de tiempo real, como cualquier otro sistema, posee dos etapas definidas: análisis y diseño.

- Análisis: en esta etapa, se especifica todo el comportamiento del sistema de tiempo real. Son impuestas las restricciones funcionales y temporales del sistema con el medio que lo rodea. Debido a la naturaleza de los sistemas de tiempo real, un aspecto fundamental es la especificación de los vencimientos del sistema para reaccionar a los estímulos externos y la periodicidad necesaria en el monitoreo de datos o en la ejecución de acciones de mantenimiento.
- Diseño: en esta etapa se determina la forma en que será organizado el sistema, para cumplir las especificaciones y para su posterior implementación. Existen diferentes alternativas para la implementación de un sistema de tiempo real duro, que compiten a la hora de elección por su eficiencia, confiabilidad o complejidad de implementación.

Ambas etapas no deben ser secuenciales o excluyentes, sino que debe existir una permanente interacción entre el analista y el diseñador para mejorar tanto las especificaciones como la implementación final. Tampoco puede definirse una delimitación precisa para cada una de las etapas si se desea obtener un buen diseño.

2. Análisis de Sistemas de Tiempo Real

2.1 El modelo del proceso

En el mundo real los sucesos ocurren en forma simultánea, por lo que el sistema que interactúa con dicho mundo debe simular un comportamiento paralelo. Esto es realizado generalmente mediante la construcción de sendos conjuntos de operaciones secuenciales, que interactúan entre ellos y con el medio en forma continua. Cada conjunto de operaciones define una

tarea y tiene como objetivo cumplir con alguna de las especificaciones determinadas para el sistema. Debido a las características de los sistemas de tiempo real, cada tarea deberá satisfacer requisitos temporales que pueden ser periódicos o no-periódicos. Esto determinará que la tarea sea periódica o no-periódica respectivamente.

Una *tarea periódica*, consiste en una computación que es ejecutada repetitivamente con un patrón regular y cíclico. La duración del intervalo entre el comienzo de una ejecución y el siguiente es denominado *período*.

Una *tarea no-periódica*, es una computación que se ejecuta como respuesta a un evento asincrónico. Es importante determinar para una tarea no-periódica el mínimo intervalo que puede existir entre dos eventos consecutivos. Este parámetro determinará la mínima separación entre los comienzos de dos ejecuciones consecutivas de dicha tarea.

La abstracción de la tarea es natural e intuitiva y permite la descomposición del problema en subproblemas. En la etapa de análisis una tarea determina el objetivo que debe lograrse para satisfacer la especificación, mientras que en la etapa de diseño significará un código de programa que debe ejecutarse para cumplir con dicho objetivo. El significado del término tarea dependerá del contexto en que sea utilizado.

Con estas consideraciones la especificación de un sistema de tiempo real constará de un conjunto de m tareas, representado por $\mathbf{S}(m)$, y definido de la siguiente forma:

$$\mathbf{S}(m) = \{ \tau_i = (T_i, D_i) \mid 1 \leq i \leq m \}$$

donde τ_i representa la tarea i sea ésta periódica o no-periódica. El parámetro T_i representa el período de una tarea periódica, o el mínimo tiempo entre ejecuciones consecutivas para tareas no-periódicas. D_i representa el vencimiento de la tarea τ_i.

Es importante notar que los parámetros T_i y D_i determinan los requerimientos temporales, que debe satisfacer la tarea τ_i para cumplir con las especificaciones de tiempo real del sistema.

2.2 Sistemas de Tiempo Real Predecibles

Generalmente el concepto de sistema de tiempo real se asocia a un sistema que debe responder muy rápidamente a los eventos. En [37] se explica lo erróneo de este concepto en los sistemas de tiempo real y se remarca la importancia de la predictibilidad antes que la velocidad. Si bien la mayor rapidez de un sistema ayudará a cumplir más fácilmente las condiciones de tiempo real, no hay que olvidar que la rapidez es un término relativo y lo importante al realizar un sistema de tiempo real es garantizar las especificaciones temporales, es decir hacer un sistema predecible.

Tal vez el mejor interrogante para aquéllos que asocian un sistema de tiempo real con un sistema rápido, es el siguiente: dado un conjunto de especificaciones de tiempo real duro implementadas, con el hardware y software más rápido posible, ¿es suficiente realizar pruebas de corrida para garantizar que todas las especificaciones serán cumplidas siempre?. La respuesta a esta pregunta es: no. Pueden hacerse grandes cantidades de pruebas al sistema, sin probar quizá la condición o combinación de condiciones en que deja de cumplir con las especificaciones. Como ejemplo, puede citarse el inconveniente que postergó el lanzamiento de nave espacial Shuttle debido a un incumplimiento de las especificaciones, que no ocurrió en las pruebas de verificación, pero sí cuando se inicializaron los equipos en la plataforma de lanzamiento [37]. Predecible y no veloz es la meta principal en un sistema de tiempo real.

Un conjunto de pruebas no garantizan que un sistema de tiempo real sea predecible puesto que, salvo en casos simples, no se puede probar exhaustivamente al sistema en un tiempo finito. Es necesario analizar el sistema antes de su implementación total, para asegurar que las especificaciones de tiempo real serán satisfechas.

3. Diseño de Sistemas de Tiempo Real

Es relativamente cierto que la mayoría de los sistemas de tiempo real son diseñados *ad hoc*. Esto no significa, sin embargo, que una aproximación científica no sea posible [37]. Una década después de esta afirmación de Stankovic, las investigaciones han obtenido resultados muy importantes que permiten modelar un sistema para validar las condiciones de tiempo real antes de su implementación. De todas maneras, el diseñador sigue enfrentándose a situaciones de compromiso, debido a que no existe una solución óptima que resuelva todas los problemas de los sistemas de tiempo real duro.

3.1 El modelo del sistema

Un típico sistema de control de tiempo real es un sistema, basado en microprocesadores, que interactúa con el medio. Los dispositivos de entrada y salida son generalmente clasificados en sensores, actuadores, registradores e interfaces. Los sensores adquieren el estado del medio y lo transforman en datos para ser procesados por el sistema. Los actuadores reciben comandos del sistema, y actúan en consecuencia para modificar ciertos parámetros del medio a controlar. Dispositivos registradores e interfaces, entre otros, permiten analizar y modificar el comportamiento del sistema para adaptarlo a diferentes condiciones de funcionamiento, según lo requiera el usuario.

Las tareas son las encargadas de relacionar todos los componentes del sistema para su correcto funcionamiento. Cuando ocurren eventos provocados por el ambiente o producidos internamente en el sistema, las tareas deben reaccionar para actuar en consecuencia antes de su respectivo vencimiento. En el diseño, las tareas estarán conformadas por software ejecutable en el hardware del sistema. En la terminología de sistemas operativos se suele denominar *procesos* a estas tareas.

Una tarea es más que el simple código de programa. Una tarea posee

además un estado de ejecución, representado por el contador de programa, y el contenido de los registros del procesador. Una tarea generalmente incluye también una pila (*stack* en inglés) para el almacenamiento temporal de datos (p.e.: parámetros pasados por las subrutinas y direcciones de retorno) y una sección de datos para el almacenamiento de las variables de la tarea.

Hay que enfatizar que el código de programa no es por sí mismo una tarea; un programa es una entidad pasiva, tal como el contenido de un archivo almacenado en disco, mientras que una tarea es una entidad activa, con un contador de programa que determina la próxima instrucción a ser ejecutada y un conjunto de recursos asociados [36]. Aunque dos tareas pueden estar asociadas al mismo programa, serán consideradas por el sistema como dos secuencias de ejecución diferentes.

Puesto que las tareas deben ser ejecutadas en un hardware con velocidad de procesamiento finita, cada una de ellas tendrá un tiempo de ejecución asociado. El máximo tiempo de ejecución de cada tarea, representado C, es un parámetro importante en el diseño de sistemas de tiempo real. De esta manera, el conjunto de m tareas, $\mathbf{S}(m)$ queda especificado de la siguiente forma para el diseñador:

$$\mathbf{S}(m) = \{ \tau_i = (T_i, C_i, D_i) \mid 1 \leq i \leq m \}.$$

Para el diseñador existen otros parámetros de la tarea como ser: la cantidad de memoria del sistema que requiere, los recursos que necesita y características especiales del hardware (procesador matemático, de red, etc.). Sin embargo, para la verificación de las especificaciones temporales, generalmente son sólo necesarios el período, el máximo tiempo de ejecución y el vencimiento de cada una de las tareas.

Mientras los parámetros T y D de las tareas dependen de las especificaciones de tiempo real descriptas en la etapa de análisis, el parámetro C dependerá de la implementación que realice el diseñador. El

máximo tiempo de ejecución dependerá de la eficiencia con que el diseñador implemente su algoritmo y de la potencia de cómputo del procesador, salvo que la tarea tenga especificaciones explícitas de su duración (por ejemplo una tarea que debe mantener encendida por 10 segundos una lámpara).

3.2 Estado de la tarea

Mientras esté activo un sistema de tiempo real, las tareas que lo componen pueden estar en uno de los siguientes estados:

- Ejecución: la tarea está siendo ejecutada por el procesador. Puede existir una sola tarea en este estado por unidad procesadora en el sistema.
- Espera: la tarea está esperando un evento para pedir ser ejecutada en el procesador. El evento puede provenir de un dispositivo de entrada/salida, un temporizador del sistema, de otra tarea o un evento interno del sistema. Pueden existir varias tareas en este estado en un instante determinado.
- Lista: la tarea está solicitando el procesador para ser ejecutada. Al igual que el estado de espera, pueden existir varias tareas en el estado de lista en un instante determinado.

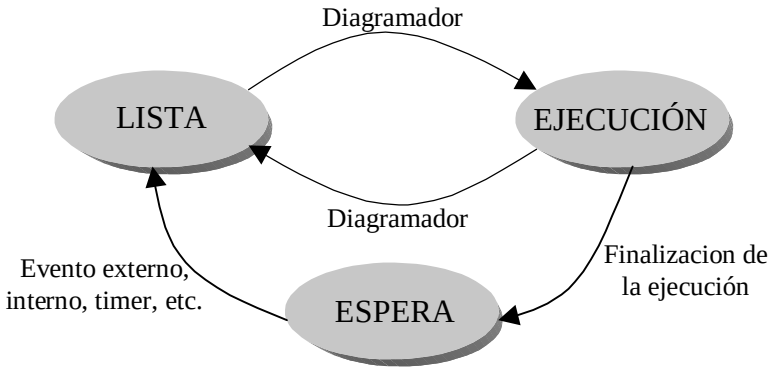


Figura 1.1

Los nombres y cantidad de los estados pueden variar según el sistema operativo del que se tratare, pero en esta tesis se utilizarán las definiciones anteriores. La figura 1.1 presenta un diagrama de estados de las tareas de

un sistema y los diferentes eventos que producen las transiciones en los estados.

Las transiciones entre los posibles estados de una tarea se deben a diferentes causas. Del estado de espera al de lista, es necesario que ocurra un evento para que suceda la transición. Cuando esta transición ocurra, se dirá que la tarea *arriba* o se *genera*. La transición del estado de ejecución al de espera ocurre cuando la tarea ha terminado de ejecutarse y ejecuta la primitiva "exit" del Sistema Operativo, para indicarle que ya no requiere más la utilización del procesador. Las transiciones del estado de lista al de ejecución y viceversa son generalmente realizadas por una tarea del Sistema Operativo denominada *diagramador de tareas* (desarrollado en la sección siguiente). El diagramador selecciona la tarea que pasará al estado de ejecución entre todas las tareas que están en el estado de lista y eventualmente pasa al estado de lista la tarea que está en estado de ejecución. Puede existir, según el sistema operativo del que se tratare, una transición que lleve una tarea del estado de lista al de espera, pero no se la considerará por no influir en los resultados de tiempo real de esta tesis.

3.3 El Sistema Operativo

Los Sistemas Operativos constituyen una parte muy importante en la mayoría de los sistemas de tiempo real. Generalmente el diseñador requiere un sistema operativo de tiempo real para una mejor administración de los recursos y para obtener un mayor nivel de abstracción en el diseño del sistema [38]. Este grado de abstracción es cada vez más necesario debido a la complejidad que poseen los ambientes que comienzan a abarcar los sistemas de tiempo real.

En la sección anterior se mencionaron los diferentes estados que pueden tener las tareas que componen el sistema y someramente la función que cumple una tarea especial denominada diagramador. De estas necesidades se desprende que en un sistema donde coexisten varias tareas,

es imprescindible llevar un registro del estado de cada una de ellas, como así también el control de las transiciones. Estas funciones, entre otras, son llevadas a cabo por el Sistema Operativo. El propósito del sistema operativo es proveer un ambiente en donde puedan ejecutarse todas las tareas y asegurar su coexistencia haciendo al sistema de cómputo conveniente para su uso. Otro objetivo es hacer un uso más eficiente y seguro del hardware del sistema.

De todas las funciones de un Sistema Operativo, la que será referenciada con mayor frecuencia, debido a la temática de esta tesis, es la que cumple el diagramador. Su función es determinar, del conjunto de tareas que están en estado de lista, la tarea que pasará al estado de ejecución.

En sistemas multiusuario, el diagramador del sistema operativo es el encargado de asegurar una ranura de tiempo del procesador a cada una de las tareas de los usuarios, para darles la sensación de que el sistema los atiende en forma exclusiva. En realidad, el sistema atiende a sólo un usuario en un instante determinado de tiempo (en sistemas monoprocesador), pero adjudica intervalos de atención a cada uno de los diferentes usuarios. En estos sistemas el principal objetivo del diagramador es asegurar que la sensación del usuario sea la de poseer un sistema a su disposición. Para esto debe garantizarle a cada uno de ellos, que el procesador del sistema le será adjudicado durante un cierto intervalo con una determinada frecuencia. Si la frecuencia es muy baja o el intervalo es muy pequeño, el usuario sentirá que el sistema se detiene frecuentemente.

En un sistema de tiempo real, el objetivo del diagramador es diferente. En estos sistemas, el diagramador debe establecer en cada instante la tarea que pasará al estado de ejecución para que ninguna de ellas pierda su vencimiento. Para esto establece, cada vez que se ejecuta el diagramador, una prioridad para cada una de las tareas del sistema. La tarea que pasará al estado de ejecución será la que posea mayor prioridad entre las que están

en estado de lista en ese instante. La prioridad de una tarea puede ser variante o invariante en el tiempo. Las reglas que utiliza el diagramador para asignar prioridades a las tareas, corresponde a una *disciplina de prioridades*. Las diversas disciplinas de prioridades, sus características y las diferencias de los diagramadores serán tratados con mayor detalle en los capítulos siguientes.

3.4 Bloque de control de la tarea (PCB)

Cada tarea es representada en el sistema operativo mediante una estructura de datos denominada *bloque de control de la tarea* (PCB del inglés process control block). En esta estructura se almacena toda la información que el sistema operativo necesita para ejecutar la tarea. Entre otros datos el PCB debe contener:

- Estado de la tarea: indica si la tarea está en estado lista, espera o ejecución.
- Contador de programa: contiene la dirección de memoria en donde se encuentra la próxima instrucción de la tarea a ser ejecutada.
- Registros del procesador: almacena el contenido de los registros del procesador, para restablecerlos cuando la tarea pase nuevamente al estado ejecución. El número y tamaño de los registros depende de la arquitectura del procesador del sistema y puede incluir acumuladores, registros de propósito general, registro de pila, etc.
- Información para la diagramación: incluye la prioridad que posee la tarea y cualquier otra información necesaria para el diagramador como ser: período de generación, próximo vencimiento, tiempo restante de ejecución, etc..
- Estado de los recursos de entrada/salida: mantiene información del estado de los recursos que posee abiertos o necesita la tarea.
- Memoria utilizada por la tarea: incluye información de los registros base y límite para el direccionamiento de memoria, tablas de página, etc.

4. Contribuciones de la tesis

Esta tesis ha sido motivada por la búsqueda de condiciones menos pesimistas que aseguren, de una manera formal, que un sistema de tiempo real cumplirá con todas las especificaciones temporales. La obtención de condiciones generales que permitan asegurar que un sistema es predecible, son muy difíciles de obtener debido a la gran complejidad que presentan las diversas características de los sistemas de tiempo real. En adelante se dirá que un sistema es diagramable para significar que el sistema es predecible y cumple todas las restricciones. También se utilizará la expresión "condiciones de diagramabilidad" para describir las condiciones que aseguran que el sistema es diagramable.

Las contribuciones de la tesis cubren tres puntos principales:

- En sistemas multitarea-monoprocesador se establecen condiciones menos pesimistas para determinar si el sistema es predecible cuando existen relaciones de precedencia entre las tareas. Esta problemática ha sido tratada anteriormente pero con disciplinas de prioridades no fácilmente implementables. En esta tesis se obtiene una condición de diagramabilidad utilizando la disciplina de prioridades por *Prioridades Fijas*, que se ha convertido en un estándar *de facto* al ser adoptada por el Departamento de Defensa de los Estados Unidos. Se demuestra también la conveniencia de convertir a una tarea independiente en un conjunto de subtareas con relaciones de precedencia, para hacer diagramable un sistema que anteriormente no lo era.
- En sistemas multitarea-multiprocesador se propone un mecanismo para evitar la pérdida de periodicidad entre las tareas que poseen relaciones de precedencia. Con este mecanismo se obtienen condiciones de diagramabilidad menos pesimistas y se comparan los resultados con otros mecanismos de diagramación. El mecanismo supone la existencia de una asignación factible resultante de la aplicación de algún método desarrollado para resolver el problema. Aunque no constituye el tema principal de esta

tesis, se describe someramente un Sistema de Producción de Aprendizaje Difuso diseñado al efecto.

- En sistemas de comunicación en tiempo real multihop por conmutación de paquetes, se halla un isomorfismo con los sistemas multitarea-multiprocesador, también en tiempo real. Esto permite trasladar los resultados y el mecanismo propuesto para sistemas multiprocesador a estos sistemas, obteniendo condiciones de diagramabilidad menos pesimistas que las existentes al momento de desarrollo de esta tesis. Se realiza una comparación con el método propuesto por Sathaye en su tesis doctoral [33].

Capítulo 2

La diagramación es la base para el buen funcionamiento de sistemas multitareas. Conmutar un procesador entre varias tares permite utilizarlo más eficientemente. En este capítulo se describe los principales conceptos de la diagramación de sistemas, como también distintas alternativas para su implementación.

1. Diagramación: principales conceptos.

El objetivo de sistemas multiprogramados es tener varias tareas ejecutándose al mismo tiempo para maximizar la utilización del o los procesadores [36]. En sistemas multiusuarios la utilidad de tiempo compartido es conmutar el procesador entre los diversos usuarios tan frecuentemente, que les permita interactuar con sus aplicaciones como si se tratase de un sistema que estuviera exclusivamente a su disposición. Contrariamente, en un sistema de tiempo real el objetivo de compartir el o los procesadores, entre las diversas tareas que lo componen, es que todas ellas cumplan con sus vencimientos.

En sistemas monoprocesador, nunca podrá haber más de una tarea ejecutándose en un determinado instante. En sistema multiprocesadores podrán ejecutarse simultáneamente tantas tareas como procesadores hubiere. Las restantes tareas del sistema estarán en estado de espera o de lista. El ente encargado de determinar la tarea que pasará al estado de ejecución de entre todas las tareas en estado de lista será denominado *diagramador* (en inglés *scheduler*). El diagramador procesa un algoritmo, denominado *algoritmo de diagramación*, para decidir cuál de las tareas que están en estado de listas pasará a estado de ejecución en el instante siguiente. En esta tesis denominaremos a la función y efecto que produce el diagramador con el término *diagramación* (en inglés scheduling).

El diagramador, generalmente implementado en el Sistema Operativo, cumple la función de administrar la utilización del procesador por las tareas que componen el sistema. Una mala administración del procesador en un sistema de tiempo compartido podría dar mucha importancia a un grupo de usuarios, relegando a un estado que impediría el uso normal del sistema a los restantes usuarios. En tiempo real, una mala o ineficiente administración del procesador podría causar la pérdida de vencimientos de algunas tareas haciendo entrar en crisis al sistema. La importante labor que cumple la diagramación en un sistema, ha motivado investigaciones que proponen diversos algoritmos que compiten en diferentes factores como ser: eficiencia, sencillez de implementación, predicción, etc.

2. Colas de estados y diagramación

El sistema operativo organiza la información del estado de cada tarea en las denominadas *colas de estados*. De esta manera existirá en la generalidad de los sistemas operativos una *cola de listas* y una *cola de espera* conformada por las tareas que están en estado de lista y espera respectivamente. En las colas de estados el sistema operativo asocia los bloques de control con las tareas que se encuentran en el estado de la cola

correspondiente. También existe un puntero que apunta a la tarea que está en estado de ejecución.

Durante el tiempo de corrida, el sistema operativo va modificando el enlace de los bloques de control de las tareas, dependiendo de la modificación del estado de las mismas. La cola donde esté ubicado el bloque de control de la tarea, dará información del estado de dicha tarea. La figura 2.1 ilustra la estructura de las colas de estados de un Sistema Operativo genérico.

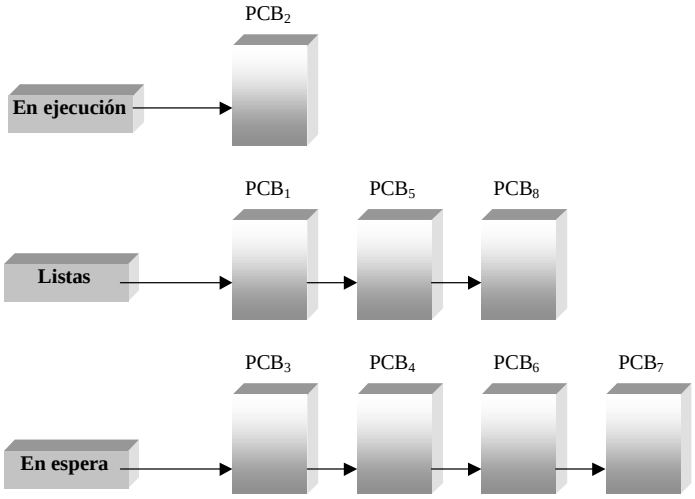


Figura 2.1

Un algoritmo de diagramación selecciona de la cola de listas la tarea que debe pasar al estado de ejecución.

3. Implementación del diagramador

Generalmente el diagramador es implementado por una tarea del Sistema Operativo. En estos casos el diagramador necesitará el procesador en forma exclusiva por un determinado tiempo, para establecer la tarea que pasará al estado de ejecución en el instante siguiente. Existen dos formas de implementar el derecho que tiene el diagramador para apoderarse del procesador:

- el diagramador espera que la tarea en estado de ejecución le ceda el procesador para poder ejecutarse.

- el diagramador interrumpe a la tarea en estado de ejecución para apoderarse del procesador.

En el primer caso el sistema es *no-apropiativo* (en inglés *non-preemptable*) mientras que en el segundo caso el sistema es *apropiativo* (en inglés *preemptable*). En esta tesis se utilizará indistintamente las expresiones *diagramación apropiativa* (*diagramación no-apropiativa*), *diagramador apropiativo* (*diagramador no-apropiativo*) o *sistema apropiativo* (*sistema no-apropiativo*).

Independientemente de que el diagramador sea apropiativo o no, los algoritmos de diagramación pueden dividirse en dos clases: los manejados por prioridades y los manejados por línea de tiempo. En los algoritmos de diagramación manejados por prioridades, no se conoce *a priori* el momento en que la tarea comenzará a ejecutarse, ni tampoco el momento en que terminará de hacerlo. En los diagramadores manejados por línea de tiempo, un cronograma es utilizado para la diagramación de las tareas. De esta manera los momentos en que cada tarea estará en estado de ejecución son conocidos anticipadamente por el diseñador.

La elección de las características del diagramador podrán estar influenciadas por particularidades del hardware donde se implemente el sistema. Una de las más importantes de estas características es la conmutación de contexto, detallada en la siguiente sección.

4. Conmutación de contexto

Conmutar el procesador que está ejecutando una tarea para pasar a ejecutar otra, requiere transferir el estado del sistema al bloque de control de la tarea que libera el procesador y cargar el estado almacenado en el bloque de control de la tarea que se apodera de él. Esta operación se denomina *conmutación de contexto*. El tiempo necesario para realizar una conmutación de contexto es una sobrecarga para el sistema, debido a que no se realiza ningún trabajo útil mientras dura dicha operación. Su

velocidad varía de hardware a hardware, dependiendo de la velocidad de la memoria, la cantidad de registros que deben ser copiados y la existencia de funciones especiales (tal como una instrucción de procesador que almacene todos sus registros). Generalmente el tiempo de conmutación de contexto varía de 1 a 1000 microsegundos.

En sistemas de tiempo real cobra significativa importancia el cálculo del máximo tiempo que lleva completar una conmutación de contexto, como también la frecuencia con que el sistema la realiza. Una conmutación de contexto muy frecuente hará ineficiente al sistema, debido a que se desperdiciará más tiempo del procesador en esta operación. En [12] se estudia, para diferentes metodologías de apropiación, la influencia de la conmutación de contexto en la eficiencia de un sistema de tiempo real.

5. Sistemas No-Apropiativos

En los sistemas no-apropiativos las tareas no son interrumpidas en su ejecución y son frecuentemente utilizados cuando la sobrecarga producida por la conmutación de contexto es alta y donde la ausencia de interrupciones hace la validación del sistema más fácil. Este tipo de mecanismo es la única posibilidad que puede utilizarse sobre determinados hardwares debido a que no utiliza recursos (por ejemplo un temporizador) necesarios en sistemas apropiativos.

Los primeros sistemas de cómputos utilizaban una diagramación por lotes que era por naturaleza no-apropiativo. Los usuarios colocaban los programas codificados en tarjetas perforadas uno a continuación del otro. La máquina procesaba todas las tarjetas perforadas de un usuario para pasar a procesar las tarjetas del siguiente usuario. De esta manera, el programa de un usuario era procesado ininterrumpidamente desde su comienzo hasta su final.

En un sistema no-apropiativo, una vez que el procesador ha sido asignado a una tarea del sistema, dicha tarea queda en control del

procesador hasta que ella misma lo ceda. La tarea cede el procesador ejecutando una llamada al Sistema Operativo cuando finaliza su ejecución o cuando permite voluntariamente que se produzca un cambio de contexto (ella pasa de estado de ejecución a estado de lista o espera).

Un sistema no-apropiativo puede ser manejado por prioridades o manejado por línea de tiempo. Entre los sistemas operativos no-apropiativos pueden citarse el MAFT [13],el Spring [39] y el Maruti [43].

6. Sistemas Apropiativos

Las tareas que se ejecutan en un sistemas apropiativo pueden ser interrumpidas durante su ejecución. Este tipo de sistema es más conveniente cuando el tiempo de conmutación de contexto es bajo o en sistemas de tiempo compartido.

El evento que motiva al diagramador a apropiarse del procesador para ejecutar el algoritmo de diagramación puede ser de diferentes naturalezas, como ser:

- la expiración de un temporizador. En este caso existe un reloj que interrumpe cada un determinado tiempo al procesador para ejecutar el algoritmo de diagramación. El tiempo entre interrupciones puede ser constante, con lo que el diagramador se ejecutará periódicamente, o variable por el propio diagramador para aumentar la utilización del procesador [12].
- la generación de una interrupción externa. El diagramador es invocado mediante un evento externo que ingresa al sistema mediante una interrupción del procesador.
- la invocación del diagramador por parte de la tarea que se está ejecutando. La tarea que está en estado de ejecución puede, a través de una llamada al Sistema Operativo, invocar al diagramador para que la utilización del procesador sea entregada a otra tarea.

La utilización de los eventos anteriores no es excluyente, pudiendo un

Sistema Operativo invocar al diagramador por una combinación arbitraria de ellos. En un sistema apropiativo puede darse el caso de que la tarea interrumpida para ejecutar el algoritmo de diagramación, sea la misma que el diagramador selecciona como próxima tarea a ejecutarse. Este caso es indeseado pues no cambia el estado del sistema por no haber una conmutación de la tarea que se ejecuta, pero provoca una pérdida de tiempo porque el procesador es utilizado para ejecutar el algoritmo de diagramación. En [12] se estudia la sobrecarga que produce evitar esa condición. Puede darse el caso de que el aumento de la complejidad del diagramador para evitar esa situación, produzca más sobrecarga que la situación misma si ésta se produce esporádicamente.

La mayoría de los Sistemas Operativos comerciales modernos son apropiativos. Como Sistemas Operativos de tiempo real pueden citarse el RT-Mach [42] y el LynxOS [20].

Desafortunadamente la diagramación apropiativa tiene su costo adicional sobre la no-apropiativa. Consideremos el caso en que dos tareas comparten datos. Una tarea, mientras está actualizando un conjunto de datos, puede ser apropiada por la segunda que comenzará a ejecutarse. Esta segunda tarea se ejecutará con el conjunto de datos inconsistentes por no estar actualizados en su totalidad, pudiendo provocar situaciones o resultados indeseados. Para evitar estos inconvenientes deben agregarse mecanismos parar coordinar el acceso a datos compartidos.

La apropiación tiene también su influencia en el aumento de la complejidad en el diseño del kernel del Sistema Operativo. Durante la ejecución de una llamada al sistema, el kernel del Sistema Operativo está realizando una operación por encargo de una tarea. Las actividades realizas por una llamada al kernel del Sistema Operativo involucran la manipulación de importantes datos del sistema (por ejemplo: configuración de dispositivos de hardware). ¿Qué sucede si la tarea que invocó la llamada al sistema es apropiada por otra tarea que requiere leer o modificar la misma

estructura de datos?. Seguramente se producirá un caos general del sistema. Algunos sistemas operativos, incluyendo varias versiones de UNIX, resuelven este problema esperando que se procesen todas las llamadas al sistema antes de realizar una conmutación de contexto. Este esquema asegura una estructura del kernel simple, pero una eficiencia en la apropiación muy pobre.

Las contribuciones originales de esta tesis versan sobre sistemas apropiativos, por lo que se desarrollarán con mayor detalle más adelante.

7. Criterios de diagramación

Diferentes algoritmos de diagramación tienen diferentes propiedades que pueden favorecer su utilización en algunos sistemas sobre otros. Existen diversos criterios que el diseñador debe considerar al elegir el algoritmo de diagramación de su sistema.

Varias características pueden ser utilizadas para comparar los algoritmos de diagramación y determinar el mejor algoritmo. Alguno de los criterios utilizados son:

- Utilización del procesador. Se desea mantener al procesador tan utilizado como fuera posible. La utilización del procesador puede variar en un rango entre 0 y 100 por ciento. En ciertos sistemas se desea que el procesador siempre ejecute alguna tarea, estando ocioso sólo cuando ninguna tarea requiera utilizarlo.
- Eficiencia de ejecución total de tareas. Una medida de la eficiencia de un sistema, es el número de tareas que son ejecutadas completamente en un determinado tiempo (en inglés *throughput*). Para sistemas con tareas extensas este parámetro puede ser, por ejemplo, de una tarea por hora, mientras que para sistemas que realizan pequeñas transacciones podría ser de decenas de tareas por segundo.
- Tiempo total de ejecución. Desde el punto de vista de una tarea, un criterio importante es el tiempo total que le llevó ejecutarse. El intervalo desde que

la tarea requiere ser ejecutada en el procesador, hasta que finaliza completamente su ejecución, es denominado tiempo total de ejecución (en inglés *turnaround time*). El tiempo total de ejecución es la suma de los intervalos que la tarea está en estado de lista y ejecución, más el tiempo necesario para acceder a los dispositivos de entrada/salida.

- Tiempo de espera. El algoritmo de diagramación no afecta el tiempo que la tarea está en estado de ejecución o esperando a un dispositivo de entrada/salida. Un criterio para evaluar un algoritmo de diagramación es la minimización de los intervalos que las tareas están en estado de lista.
- Tiempo de respuesta. En un sistema interactivo, el tiempo total de ejecución puede ser un mal criterio de diagramación. Frecuentemente, las tareas pueden brindar resultados intermedios que le sirven al usuario para observar la tendencia de los cómputos que realiza. En estos casos es bueno que los primeros resultados sean producidos lo más tempranamente posible. Esta medida, denominada *tiempo de respuesta*, es el tiempo en que el usuario tiene la primera salida del sistema.

Generalmente los sistemas operativos desean maximizar la utilización del procesador y la eficiencia de ejecución total de tareas y minimizar el tiempo total de ejecución, el tiempo de espera y el tiempo de respuesta. En la mayoría de los sistemas se desea optimizar el valor promedio de cada uno de estos tiempos. Sin embargo, existen circunstancias en que se desea optimizar el valor mínimo o máximo de un parámetro, más que su valor promedio. Este es el caso de los sistemas de tiempo real en donde se desea minimizar el máximo tiempo total de ejecución de una tarea. Existe un mal concepto que afirma que un sistema de tiempo real debe ser rápido, haciendo referencia a que debe tener un bajo valor de su tiempo de respuesta. Sin embargo en un sistema de tiempo real es necesario *predecir* el máximo tiempo total de ejecución de una tarea. En un sistema de tiempo real duro no importa cuán antes de su vencimiento termina de ejecutarse una tarea, sino asegurar que la tarea se ejecutará completamente antes de

vencimiento.

También ha sido sugerido que, para sistemas interactivos con el usuario (tal como un sistema de tiempo compartido), es más importante minimizar la varianza que el promedio en el tiempo de respuesta. Un sistema con "razonable" y constante tiempo de respuesta, es más deseable que uno con menor promedio pero más variación.

8. Algoritmos de diagramación

Los algoritmos de diagramación deben decidir a cuál de las tareas en la cola de listas le será asignado el procesador para ejecutarse. Existen diferentes algoritmos de diagramación que serán descriptos muy brevemente en esta sección. Se prestará principal importancia a las características temporales de los mismos.

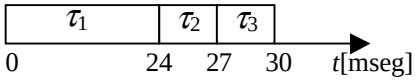
8.1 Algoritmo de diagramación "Primera en llegar primera en ejecutarse" (First-Come, First Served Scheduling).

Decididamente el más simple de los algoritmos de diagramación es el algoritmo primera en llegar-primera en ejecutarse (*FCFS*) [36]. Con este esquema la primera tarea que requiera el procesador será la primera en obtenerlo. La implementación de un algoritmo de este tipo es fácil mediante una pila primera en llegar-primera en ejecutarse (First In First Out, FIFO). La tarea, junto con su PCB asociado, pasa al estado de lista y es ubicada al final de la cola de listas. Cuando el procesador ha terminado de ejecutar la tarea en estado de ejecución, el diagramador selecciona la primera tarea de la cola de lista para asignárselo. Las primeras máquinas de cómputos que utilizaban tarjetas perforadas implementaban este tipo de diagramación. El código de un algoritmo de diagramación FCFS es simple de entender e implementar.

Sin embargo el promedio del tiempo de espera de este algoritmo es frecuentemente alto. Consideremos un conjunto de tareas que pasan al estado de listas en el instante t=0, con el tiempo de procesamiento para su ejecución completa (C) en milisegundos:

Tarea	C_i
τ_1	24
τ_2	3
τ_3	3

Si las tareas arriban a la cola de listas con el orden τ_1, τ_2, τ_3, el algoritmo hará que se ejecuten con esa misma disposición. La utilización del procesador es representada por un diagrama de tiempo en la siguiente figura.



El tiempo de espera es 0 milisegundos para la tarea τ_1, 24 milisegundos para la tarea τ_2 y 27 milisegundos para la tarea τ_3. De esta forma el tiempo promedio de espera es (0+24+27)/3 = 17 milisegundos. Si las tareas arriban con orden τ_2, τ_3, τ_1, el promedio del tiempo de espera será (6+0+3)/3 = 3 milisegundos. Así se ve que el promedio del tiempo de espera es no mínimo y varía substancialmente si el tiempo total de ejecución de las tareas son considerablemente diferentes.

El algoritmo FCFS es no-apropiativo. Una vez que el procesador ha sido asignado, la tarea se ejecuta hasta que lo libere. La liberación ocurre cuando la tarea finaliza completamente su ejecución o pasa al estado de espera, aguardando la respuesta de un dispositivo de entrada/salida.

8.2 Algoritmo de diagramación "La tarea más rápida primero" (Shortest-Task First Scheduling).

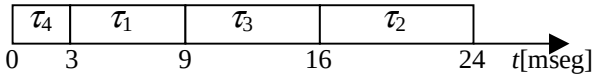
Un algoritmo de diagramación diferente es el de "la tarea más rápida primero" (STF). En este algoritmo existe un peso, asociado a cada tarea, directamente proporcional a su mayor tiempo de ejecución. Cuando el

procesador está disponible, el diagramador selecciona la tarea en estado de lista con menor peso asociado. Si existen más de una tarea con el mismo peso, el algoritmo FCFS es utilizado para desempatar entre ellas.

Consideremos un sistema integrado por el siguiente conjunto de tareas con los tiempos de ejecución en milisegundos:

Tarea	C_i
τ_1	6
τ_2	8
τ_3	7
τ_4	3

Si el instante t=0 todas las tareas pasan al estado de lista, el diagrama de tiempo de utilización del procesador será el siguiente:



El tiempo de espera para la tarea τ_1 es de 3 milisegundos, para la tarea τ_2 es 16 milisegundos, para la tarea τ_3 es 9 milisegundos y para la tarea τ_4 es 0 milisegundos. Así el promedio del tiempo de espera es (3+16+9+0)/4 = 7 milisegundos. El promedio del tiempo de espera es menor, a lo sumo igual (caso en que las tareas pasan a estado de lista ordenadas por sus tiempos de ejecución), al promedio del tiempo de ejecución del algoritmo FCFS.

La dificultad del algoritmo STF es la asignación de pesos a las tareas. El programador debe calcular el tiempo que le demandará la ejecución completa de la tarea para poder asignarle el peso correspondiente.

El algoritmo STF puede ser implementado tanto en un diagramador apropiativo como en un diagramador no-apropiativo. Diagramadores basados en el vencimiento, con muy buenas características de tiempo real, son variaciones apropiativas de este algoritmo. Estos diagramadores son "Menor Tiempo al Vencimiento" y "Menor Tiempo de Latencia" que serán desarrollados en capítulos siguientes.

8.3 Algoritmo de diagramación por prioridades.

El algoritmo de diagramación STF es un caso de algoritmo de diagramación por prioridades. En este tipo de diagramación, una prioridad es asociada a cada una de las tareas del sistema y el procesador es asignado a la tarea en estado de lista que posea mayor prioridad. Cuando las tareas poseen igual prioridad, un criterio de desempate es aplicado sobre ellas, por ejemplo FCFS.

Las prioridades que pueden ser asignadas a las tareas generalmente son un rango fijo de números, tal como de 0 a 255 (POSIX 4), o de 0 a 4095. Sin embargo, no existe convención si 0 es la mayor o la menor prioridad. Algunos sistemas asocian los números de menor rango a bajas prioridades mientras que otros sistemas lo asocian a altas prioridades.

Las prioridades pueden ser definidas interna o externamente. En las prioridades definidas internamente alguna medida cuantitativa o cualitativa es utilizada para la asignar la prioridad de la tarea. Por ejemplo pueden ser utilizadas el máximo tiempo de ejecución de la tarea, la cantidad de memoria requerida, o el número de archivos abiertos. En la definición de prioridades externas es utilizado un criterio externo al sistema, por ejemplo importancia de la tarea, para asignar las prioridades.

La diagramación por prioridades puede ser apropiativa o no-apropiativa. Cuando un tarea pasa al estado de lista, su prioridad es comparada con la prioridad de la tarea que en ese momento está en estado de ejecución. En sistemas apropiativos, el diagramador producirá un cambio de contexto si la prioridad de la tarea colocada en la cola de lista es mayor que la prioridad de la tarea en estado de ejecución. En un sistema no-apropiativo con diagramación por prioridades, la tarea será simplemente colocada al comienzo de la cola de listas.

La diagramación por prioridades tiene características muy adecuadas para sistemas de tiempo real, y debido a que los resultados originales de esta tesis están basados en la utilización de un caso particular, denominado

Prioridades Fijas, un análisis más profundo será realizado en los capítulos siguientes.

8.4 Algoritmo de diagramación "Rueda Cíclica" (Round Robin Scheduling).

La diagramación por *Rueda Cíclica* (RR) fue diseñada para sistemas de tiempo compartido. Es similar al algoritmo de diagramación FCFS, pero se adiciona apropiación para conmutar de contexto entre las tareas. Se define un pequeño intervalo de tiempo, denominado *ranura*, que a menudo sirve también como unidad de tiempo. En este diagramador existe un temporizador que interrumpe al procesador una vez por ranura. El algoritmo de diagramación se ejecuta en cada interrupción, produciendo en consecuencia una conmutación de contexto cada ranura. En cada apropiación la tarea que estaba en estado de ejecución pasa al final de la cola de listas y la primera tarea en la cola es seleccionada para ejecutarse en el procesador. Cuando una tarea pasa del estado de espera al de lista, su PCB es ubicado al final de la cola de listas.

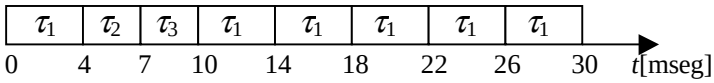
El algoritmo de diagramación se ejecutará a intervalos regulares de una ranura, excepto cuando la tarea que está en estado de ejecución libera al procesador antes de la interrupción del temporizador.

El promedio del tiempo de espera de este tipo de algoritmo es frecuentemente grande y depende de la duración de la ranura. Consideremos un sistema integrado por el siguiente conjunto de tareas con los tiempos de ejecución en milisegundos:

Tarea	C_i
τ_1	24
τ_2	3
τ_3	3

Todas las tareas pasan al estado de lista en el instante t=0, siendo ordenadas τ_1, τ_2 y τ_3 en la cola de listas. Supongamos que el sistema tiene configurada su ranura en 4 milisegundos. Un diagrama de tiempo de la

utilización del procesador es mostrado en la siguiente figura.



Nota: el tiempo utilizado por el diagramador no se representa en este diagrama por considerarse mucho menor que el intervalo de una ranura.

Se observa que en un algoritmo de diagramación por rueda cíclica, ninguna tarea está en estado de ejecución por más de una ranura excepto que sea la única tarea que requiera el procesador. En este último caso existe una sobrecarga en la utilización por realizar una conmutación de contexto innecesaria.

Si existen m tareas en la cola de listas y la duración de una ranura es q, cada tarea utilizará el procesador una ranura en cada intervalo de $m \times q$ y tendrá que esperar $(m\text{-}1) \times q$ hasta la próxima ranura en que tenga disponible el procesador.

La eficiencia de este algoritmo de diagramación depende de la cantidad de tareas en estado de lista y del intervalo en que sea configurada la ranura. En un extremo, si la duración de ranura es infinito, el algoritmo RR se comporta como FCFS. Si el intervalo es muy pequeño, la conmutación de contexto reducirá la eficiencia en la utilización del procesador por las tareas del sistema.

9. Diagramación de Sistemas Multiprocesador.

Hasta este punto, en los algoritmos de diagramación descriptos, se consideró que existen varias tareas que requieren el uso de un procesador. Si existen varios procesadores, el problema de diagramación se hace más complejo. Varias posibilidades han sido propuestas y, como en sistemas con un solo procesador, no existe una solución óptima para todos los criterios de diagramación.

Existen dos esquemas básicos para la construcción de sistemas

multiprocesador. Diremos que un sistema multiprocesador es de *procesamiento paralelo* (fuertemente intercomunicados) cuando los procesadores comparten memoria y un reloj común. Por el contrario, diremos que un sistema multiprocesador es de *procesamiento distribuido* (débilmente intercomunicado) cuando los procesadores no comparten memoria ni un reloj común. En este último caso los procesadores se comunican entre ellos a través de diferentes tipos de enlaces (p.e. redes de comunicación, líneas telefónicas, radioenlaces, etc.).

Las características de los procesadores en un sistema multiprocesador pueden ser iguales o distintas. En el primer caso diremos que el sistema es *homogéneo*, en tanto que en el segundo diremos que el sistema es *heterogéneo*. En los sistemas homogéneos cualquier tarea puede ejecutarse en cualquier procesador, mientras que en los sistemas heterogéneos una tarea puede ejecutarse sólo en un subconjunto de los procesadores del sistema.

En un sistema monoprocesador la administración del único procesador del sistema se lleva a cabo principalmente mediante las colas de estados y el algoritmo de diagramación. Todos estos componentes están dedicados al único procesador existente. En un sistema multiprocesador existen diferentes alternativas para implementar dichos componentes. Puede existir una sola implementación de las colas de estados y del algoritmo de diagramación para todos los procesadores que integran el sistema. En este método, denominado de *diagramador global*, las tareas que requieren un procesador para ejecutarse son colocadas en la única cola de listas del sistema. El algoritmo de diagramación debe seleccionar la tarea que pasará a estado de ejecución, pero además el procesador que le corresponderá. En estos sistemas existe un *procesador maestro* que ejecuta el algoritmo de diagramación y mantiene las colas de estados, y un conjunto de *procesadores esclavos* que tienen la única función de ejecutar las tareas que le indica el procesador maestro. Debido a que todos los procesadores no

realizan las mismas funciones, estos sistemas se denominan de *multiprocesamiento asimétrico*.

Por otra parte, en los *sistemas de multiprocesamiento simétrico* un algoritmo de diagramación es implementado en cada uno de los procesadores. Existe una única implementación de las colas de estados que es accedida por todos los algoritmos de diagramación. En este tipo de configuración la complejidad del diagramador es mucho mayor que en los sistemas asimétricos debido a la necesidad de sincronizar el acceso a las colas de estados.

En los sistemas con diagramación global, el diagramador debe comprobar que el procesador dispone de todos los recursos necesarios para la completa ejecución de la tarea seleccionada.

En los sistemas multiprocesador con *diagramación local*, cada procesador tiene su propio algoritmo de diagramación y sus propias colas de estados. El subconjunto de tareas que se ejecutarán en un procesador son predeterminadas en el momento de diseño o de inicialización del sistema, o se asignan dinámicamente en tiempo de corrida a través de un algoritmo de diagramación largo. La diagramación local es la opción más adecuada para sistemas multiprocesador débilmente interconectados.

En adelante cuando se haga referencia a sistemas multiprocesadores se referirá, salvo que se indique expresamente lo contrario, a sistemas de procesamiento distribuido con diagramación local.

Capítulo 3

En este capítulo se desarrollan las principales nociones sobre condiciones necesarias y/o suficientes para definir el concepto de *pesimismo* de una condición. Se describe también los principales algoritmos de diagramación utilizados en sistemas de tiempo real, desarrollando sus características más importantes.

1. Introducción al Análisis de la Diagramabilidad

El análisis de la diagramabilidad en sistemas de tiempo real permite garantizar que los requerimientos temporales serán satisfechos. La siguiente sección describe los trabajos de investigación más relevantes hechos sobre sistemas apropiativos y no apropiativos. En [28] y [40] se describen resúmenes de los principales resultados obtenidos sobre diagramación en sistemas de tiempo real.

En el capítulo anterior se desarrollaron diferentes alternativas para la implementación del diagramador del sistema. Se propusieron distintos criterios de selección y medición según su comportamiento y eficiencia. De esta manera, por ejemplo, se puede desear que el diagramador de un

sistema multiusuario tenga un bajo promedio y baja varianza del tiempo de respuesta. En un sistema de tiempo real duro, es necesario que el diagramador garantice una cota superior del tiempo total de ejecución de cada una de las tareas que lo integran. Esto se debe a que en un sistema de tiempo real duro no importa si la tarea se ejecuta completamente mucho antes de su vencimiento sino que simplemente lo haga *antes* de su vencimiento.

Debido a la naturaleza de los sistemas de tiempo real, no es suficiente simular su comportamiento para garantizar que todos los requerimientos temporales serán satisfechos. La simulación probaría a lo sumo la presencia de errores con respecto a la especificación, no su ausencia. El análisis de la diagramabilidad es la base teórica para comprobar que las especificaciones temporales serán satisfechas siempre por el sistema de tiempo real. Este análisis debe realizarse antes de la implementación y puesta en funcionamiento del sistema.

2. Condiciones Necesarias, Suficientes y Pesimismo

Debido a la gran variedad de configuraciones que puede tener un sistema de tiempo real, diversos métodos de análisis han sido desarrollados para cada caso. Las condiciones de diagramabilidad que establecen estos métodos pueden ser *necesarias*, *suficientes* o *suficientes y necesarias* (*exactas* para algunos autores).

2.1 Condición Necesaria

Si un evento, *E*, no ocurre en ausencia de una circunstancia, *C*, entonces *C* es una *condición necesaria* para la ocurrencia de *E*. Una condición necesaria, en otras palabras, es una condición que debe ocurrir si el fenómeno del cual ella es la *causa* se produce. Si *X* es condición necesaria para que el fenómeno *Y* se produzca, entonces *Y* nunca se

producirá si *X* no ocurre. La presencia de oxígeno, por ejemplo, es una condición necesaria para que se produzca el fuego. Esto es porque sin oxígeno es imposible que exista fuego. Una forma de notación de una condición necesaria es:

Si *no C*, entonces *no E*.

Ejemplo:

Si *una persona es menor de 18 años*, entonces *no puede votar*.

De esta manera es condición necesaria ser mayor de 18 años para poder votar.

En tiempo real, un método de análisis necesario garantiza que el sistema es no-diagramable si no se satisfacen las condiciones, pero no asegura su diagramabilidad en caso contrario.

2.2 Condición Suficiente

Si en evento, *E*, siempre ocurre en presencia de una circunstancia, *C*, entonces *C* es una *condición suficiente* para la ocurrencia de *E*. Una condición suficiente para la ocurrencia de un evento, en otras palabras, es una circunstancia en cuya presencia, el evento siempre ocurre. Una condición suficiente siempre produce el evento del cual es *causa*. Si *X* es una condición suficiente de *Y*, entonces siempre que se produzca *X* ocurrirá *Y*. Una forma de notación de una condición suficiente es:

Si *C*, entonces *E*.

Ejemplo:

Si *llueve* entonces *la calle se moja*.

De esta manera es condición suficiente que llueva para que la calle se moje.

En tiempo real, un método de análisis suficiente garantiza que si las condiciones del método son satisfechas el sistema es diagramable pero, en el caso de que no se satisfagan las condiciones, no se puede asegurar que el sistema sea no-diagramable.

La presencia de oxígeno es condición necesaria pero no suficiente para que se produzca fuego: esto es porque la presencia de oxígeno no garantiza la ocurrencia de fuego. Sin embargo, la presencia de oxígeno junto con un material combustible y alta temperatura garantizan la ocurrencia de fuego. De esta manera la presencia combinada de oxígeno Y un material combustible Y alta temperatura es una condición suficiente para que se produzca el fuego. Si se da esta condición, el fuego siempre se produce. Por otra parte, si una condición de suficiencia no se cumple, no se puede determinar si no ocurre el fenómeno. Consideremos la proposición "si llueve, entonces la calle está mojada". Que llueva es condición suficiente para que la calle esté mojada, pero no es una condición necesaria. Otras circunstancias pueden mojar la calle (por ejemplo, la rotura de un conducto de agua, o el tránsito de un camión hidrante). Por lo tanto, la proposición "si *C*, entonces *E*" no implica la proposición "si *no C*, entonces *no E*".

Finalmente, una condición puede ser *necesaria y suficiente* para la ocurrencia de un evento. En tal caso, si *X* es condición necesaria y suficiente para la ocurrencia de *Y*, entonces *Y* nunca podría ocurrir si no se produce *X* (condición necesaria); y si *X* se produce entonces *Y* ocurre (condición suficiente). En sistemas de tiempo real, los métodos de análisis de diagramabilidad necesarios y suficientes pueden determinar, mediante sus condiciones, si el sistema será, o no, diagramable.

2.3 Pesimismo

Existen diferentes métodos para analizar la diagramabilidad de sistemas de tiempo real de distintas características. Cada método establece condiciones que aseguran la diagramabilidad, o no, de un sistema. Para comparar diferentes métodos se utiliza el concepto de *pesimismo*.

Supongamos que el conjunto universal contiene todas las alternativas que puede tener un sistema de tiempo real. De este modo cada elemento del conjunto universal determinará un conjunto de tareas con sus respectivos

períodos, tiempos de ejecución máximos y vencimientos. Para simplificar el desarrollo del concepto de pesimismo, se considerará que estos datos especifican todos los sistemas de tiempo real duro posibles.

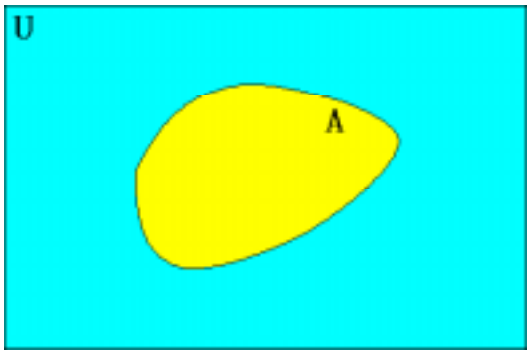


Figura 3.1

De todo el conjunto universal, sólo un subconjunto de sus elementos serán diagramables en un sistema monoprocesador mediante la utilización de un determinado diagramador. En la figura 3.1 se usa un diagrama para representar el conjunto Universal (**U**) y el subconjunto de sistemas de tiempo real diagramables (**A**).

Una condición suficiente de diagramabilidad determina un subconjunto **B** integrado por elementos que pertenecen al subconjunto **A** (**B**$\subseteq$**A**). En la figura 3.2 se observa que una condición suficiente determina un subconjunto de sistemas de tiempo real diagramables, pero no los abarca completamente.

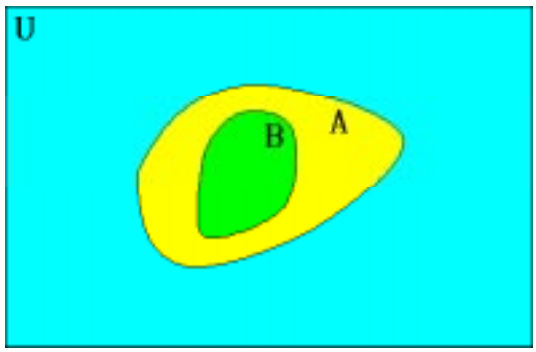


Figura 3.2

Como puede observarse, existen elementos del conjunto de sistemas diagramables **A**, que no pertenecen al subconjunto **B** que determina la condición suficiente. Debido a esto no se puede asegurar que un sistema

será no-diagramable por no cumplir una condición suficiente, pues existen sistemas diagramables perteneciente a **A-B** que no cumplen dicha condición, pero sin embargo son diagramables. Cuanto menor sea el subconjunto **A-B**, mejor será la condición de suficiencia. En la figura 3.3 puede observarse que la condición suficiente que produce el subconjunto **B'**, permite asegurar la diagramabilidad de una mayor cantidad de sistemas. En sistemas de tiempo real diremos que la condición suficiente que produce el subconjunto **B'** es *menos pesimista* que la que produce el subconjunto **B**.

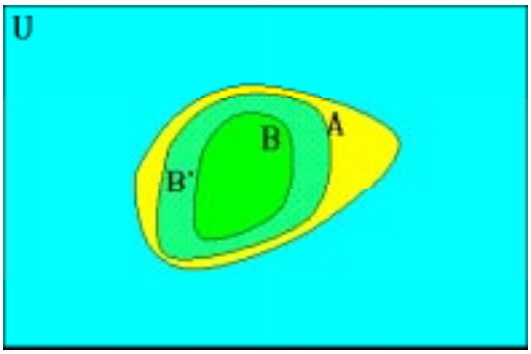


Figura 3.3

Por otro lado, una condición necesaria de diagramabilidad determina un subconjunto **C**, integrado por elementos que pertenecen al conjunto universal pero contiene completamente al subconjunto de soluciones **A** (**A**⊆C). En la figura 3.4 se observa que una condición necesaria determina un subconjunto de sistemas de tiempo real que contiene completamente a los sistemas diagramables.

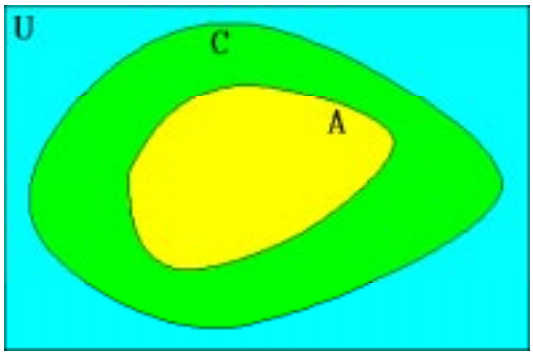


Figura 3.4

Se observa que si el sistema no pertenece a **C**, será no diagramable debido a que todo el subconjunto de sistemas diagramables está contenido

por **C**. Sin embargo, **C** está compuesto por los sistemas diagramables pertenecientes al subconjunto **A** más los sistemas no diagramables pertenecientes a **C-A**. Esto imposibilita asegurar la diagramabilidad de un sistema que pertenece al conjunto **C**. Cuanto más elementos pertenezcan al subconjunto **C-A**, menos certeza tendrá la condición necesaria en el sentido de que existirán más sistemas no diagramables que la condición no podrá determinar. En la figura 3.5 se observa que la condición necesaria que determina el subconjunto **C'** permite asegurar la no-diagramabilidad de más sistemas. En sistemas de tiempo real diremos que la condición necesaria que produce el subconjunto **C'** es *menos pesimista* que la que produce el subconjunto **C**.

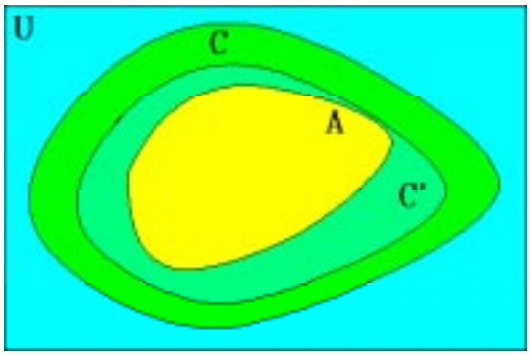


Figura 3.5

Se observa que las condiciones menos pesimistas permiten asegurar la diagramabilidad, o no, de una mayor cantidad de sistemas al disminuir el número de elementos pertenecientes a los subconjuntos **A-B** y **C-A** para las condiciones suficientes y las condiciones necesarias, respectivamente. En el límite, cuando **A-B** = ∅ o **C-A** = ∅, nos encontramos con una condición necesaria y suficiente y el subconjunto que determina está integrado por todos los elementos pertenecientes a **A**.

2.4 Condiciones y Análisis de Diagramabilidad

Se puede creer que las condiciones necesarias y las condiciones suficientes no son útiles cuando se ha encontrado la expresión de la condición necesaria y suficiente. Esto en cierta medida es verdadero pero

debe considerarse también la complejidad que involucra la resolución de una condición.

Puede darse la situación en la que el cálculo de una condición necesaria y suficiente involucra un costo computacional elevado, mientras que una condición suficiente (necesaria) permite asegurar la diagramabilidad (no diagramabilidad) del sistema más fácilmente. Una expresión sencilla puede ser utilizada en un sistema que desea determinar su diagramabilidad en tiempo de ejecución. Si la condición de diagramabilidad necesaria y suficiente es tan compleja que requiere la utilización del procesador en forma exclusiva, seguramente su cálculo será la causa de que el sistema entre en crisis. Por otro lado, una condición suficiente, o necesaria, fácilmente calculable, permitirá tomar acciones rápidamente cuando no se pueda asegurar la diagramabilidad del sistema, sin producir demasiada sobrecarga por su cálculo.

Cuando, por la complejidad del sistema, no se puede hallar la condición necesaria y suficiente que determina su diagramabilidad, se desean obtener condiciones suficientes y condiciones necesarias lo menos pesimistas posibles.

Por otro lado, la intersección de condiciones necesarias o la unión de condiciones suficientes permiten obtener condiciones menos pesimistas. En la figura 3.6 puede observarse que la combinación produce condiciones menos pesimistas que las condiciones originales individualmente.

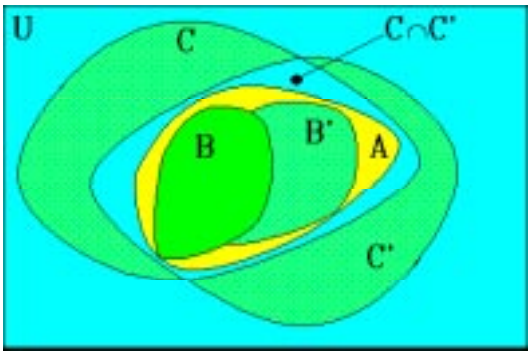


Figura 3.6

3. Diagramación Estática y Dinámica.

Los algoritmos de diagramación pueden ser divididos en dos grandes clases: estáticos o dinámicos. Diagramación estática se refiere, en el contexto de esta tesis, al hecho de que el algoritmo de diagramación tiene un completo conocimiento del conjunto de tareas y sus características tales como: vencimientos, tiempos de computación, relaciones de precedencia y futuros tiempos de generación. Estas consideraciones son verdaderas para algunos sistemas de tiempo real. En este trabajo se utilizará la definición de [40]: "si todos los futuros tiempos de generación de las tareas son conocidos cuando el algoritmo es desarrollado, entonces el diagramador es estático". Los primeros sistemas de tiempo real utilizaban el algoritmo de diagramación estático denominado *ejecutivo cíclico* (descripto más adelante).

Por otro lado, un algoritmo de diagramación dinámico (en el contexto de esta tesis) tiene un completo conocimiento de las tareas generadas, pero desconoce el instante en que se producirán nuevas generaciones de las tareas del sistema.

En la construcción de sistemas de tiempo real, un análisis de la diagramabilidad *a priori* debe ser hecho independientemente de que el algoritmo de diagramación sea dinámico o estático. En algunos sistemas de tiempo real, se pueden identificar el conjunto máximo de tareas con sus peores requerimientos, y aplicar un algoritmo de diagramación estático para producir una diagramación estática. Si todas las consideraciones efectuadas en el análisis de diagramabilidad se mantienen en tiempo de ejecución, todas las tareas cumplirán con sus vencimientos.

En los casos en que no pueden conocerse en el momento de análisis todas las características de las tareas del sistema de tiempo real, debe realizarse el análisis considerando la utilización de un algoritmo de diagramación dinámico.

4. Algoritmos de diagramación en tiempo real

Se ha afirmado que un sistema de tiempo real debe ser predecible. Para asegurar esto, debe analizarse la diagramabilidad del algoritmo utilizado. Es necesario conocer en la etapa de diseño, que el algoritmo de diagramación permitirá, a cada una de las tareas que integran el sistema, cumplir con sus respectivos vencimientos.

En la evolución de las investigaciones de sistemas de tiempo real han sido propuestos diferentes algoritmos de programación. Para cada uno de ellos se han desarrollado las condiciones que aseguran la diagramabilidad del sistema. A continuación se describirán los principales algoritmos de diagramación utilizados en sistemas de tiempo real.

4.1 Algoritmos de diagramación estáticos

4.1.1 Ejecutivo Cíclico

El algoritmo de diagramación ejecutivo cíclico fue utilizado en los primeros sistemas de tiempo real. La figura 3.7 ilustra el algoritmo de diagramación ejecutivo cíclico. Este algoritmo ejecuta las tareas en una secuencia de subtareas no apropiativas, volviendo a ejecutar cada una de ellas en un orden fijo durante todo el tiempo de corrida del sistema.

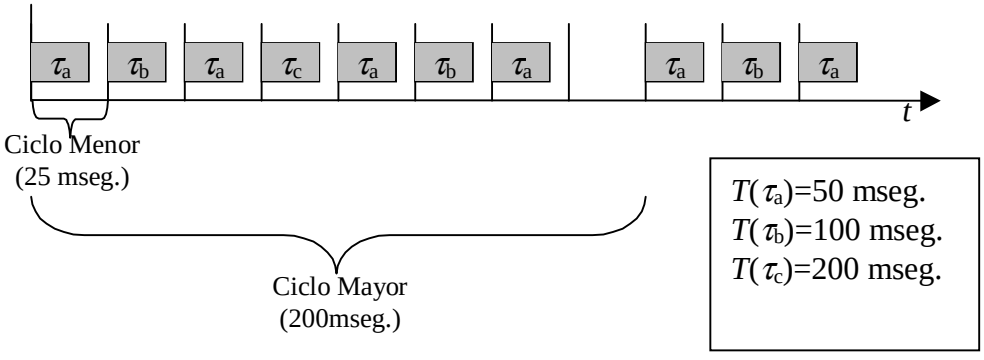


Figura 3.7

Las tareas son ejecutadas en el procesador de acuerdo a una lista que posee el algoritmo de diagramación desde el instante inicial y que

permanece invariante. Cuando la última tarea de la lista es ejecutada, el algoritmo comienza nuevamente por la que se encuentra al comienzo. Puesto que deben conocerse los futuros tiempos de generación de todas las tareas, este algoritmo de diagramación es efectivamente estático. El período en que se ejecutan todas las tareas de la lista se denomina *ciclo,* o *ciclo mayor* en el caso en que las tareas no necesiten ejecutarse con la misma frecuencia. Cuando las frecuencias no son idénticas, en la lista de tareas se define una secuencia tal que cada tarea es repetida adecuadamente para cumplir sus requerimientos temporales. En este caso las tareas serán ejecutadas individualmente en un ciclo menor, y el ciclo mayor será el mínimo común múltiplo de las frecuencias de las tareas [3].

El período de cada ciclo menor es generalmente configurado con un valor adecuado para facilitar la implementación. Mientras el ciclo mayor es producido por la longitud de la lista de tareas del diagramador, el ciclo menor puede ser originado por una interrupción periódica de un temporizador u otra interrupción que mantiene la sincronización del sistema con el ambiente externo. De esta manera, en cada activación de la interrupción, el diagramador otorga la utilización del procesador a la siguiente tarea de la lista. Las tareas deben ser ordenadas en la lista de tal forma que todas cumplan con los requerimientos de tiempo real.

Cuando una tarea finaliza su ejecución realiza una llamada al sistema y éste queda en un ciclo de espera hasta el comienzo del próximo ciclo menor. Una sobrecarga del sistema puede ser detectada al comprobar que el procesador no está en el estado de espera al comienzo de un ciclo menor. La manipulación de un estado de sobrecarga es muy difícil en un sistema con diagramación por ejecutivo cíclico. Las acciones que pueden realizarse en esta situación son: 1) permitir a la tarea finalizar su ejecución, lo cual puede provocar una desincronización del resto de las tareas del sistema o, 2) dejar inconclusa la ejecución de la tarea e indicar al usuario la sobrecarga del sistema. Ambas acciones son muy peligrosas pues pueden

hacer que el sistema entre en crisis.

4.1.1.1 Ventajas del algoritmo de diagramación ejecutivo cíclico

1. Asumiendo que no existe sobrecarga, se puede asegurar la diagramabilidad del sistema en todo momento. Esto significa que es posible predecir el comportamiento del sistema luego de su instante inicial. Así, asumiendo que las especificaciones temporales del ambiente externo están correctamente determinadas, es claro que todas las condiciones de tiempo real serán satisfechas.
2. Teniendo en cuenta que las tareas individualmente no pueden apropiarse del procesador en cualquier instante arbitrario, la estructura del sistema puede simplificarse. De esta forma, el diseñador confecciona la lista de tareas del ejecutivo cíclico para evitar el uso de mecanismos para compartir recursos (p.e., semáforos, rendezvous, monitores).
3. Considerando la estructura del sistema, se puede observar que el diseñador puede lograr una muy baja sobrecarga producida por los cambios de contexto. Para esto, debe establecerse una adecuada lista de tareas del diagramador.
4. Observando el algoritmo de diagramación por ejecutivo cíclico se concluye que su implementación es muy sencilla lo que mejora su confiabilidad.

4.1.1.2 Desventajas del algoritmo de diagramación ejecutivo cíclico

1. La más seria de las desventajas del algoritmo ejecutivo cíclico es la fragilidad de su aplicación. Por "fragilidad" se quiere expresar que cualquier cambio en el sistema, tanto en la etapa de implementación o en la etapa de mantenimiento, generará un nuevo y completo análisis de la diagramabilidad del sistema. Pequeños cambios en las tareas que integran el sistema, pueden requerir una reconfiguración total de la lista de tareas del diagramador, con su correspondiente análisis de diagramabilidad.
2. La integración en el tiempo de pequeños errores en el temporizado del diagramador podrá hacer entrar en crisis a todo el sistema. Errores en el

cálculo del tiempo de ejecución de alguna tarea del sistema, o en la frecuencia del temporizador, producirá que, eventualmente, ninguno de los requerimientos temporales sean satisfechos. Esto significa que el diagramador no permite una suave degradación ante una de estas fallas.

3. El análisis de la diagramabilidad e implementación del sistema aumentan en complejidad cuando existen tareas con grandes diferencias en sus períodos de generación y tiempos de ejecución. Una tarea con elevado tiempo de ejecución deberá dividirse en varias subtareas si debe coexistir con tareas que poseen un período de generación pequeño. Esto produce un incremento en los cambios de contextos que debe realizar el diagramador, y aumenta la complejidad en la implementación pues debe analizarse la consistencia de los recursos compartidos.

4.2 Algoritmos de diagramación dinámicos y disciplinas de prioridades.

En un sistema de tiempo real multitarea-monoprocesador un conjunto de *m* tareas requiere el procesador para ejecutarse. En sistemas en donde no se conocen los instantes en que se producirán las generaciones de las tareas, son necesarios los algoritmos de diagramación dinámicos. La determinación de la tarea que tendrá derecho a la utilización del procesador es realizada por el diagramador, que selecciona una de las tareas que en ese momento se encuentran en estado de lista. Para esta selección, el diagramador implementa una relación de orden denominada *disciplina de prioridades*, que notaremos R. *x*R*y* significa que la tarea *x* tiene mayor privilegio sobre la tarea *y* en el derecho de utilización del procesador; el conjunto de tareas ordenados por la relación R se denomina *pila de prioridades*. El procesador es otorgado, por un intervalo de tiempo denominado *ranura*, a la tarea que, estando en la cola de listas, tiene mayor prioridad. Según la implementación del diagramador, el tiempo de ranura puede ser fijo o variable. Al comienzo de cada ranura, el diagramador

otorga el procesador a la tarea de mayor prioridad que lo requiera.

A continuación se describen las disciplinas de diagramación dinámica de uso más frecuente, todas las cuales admiten variantes y combinaciones.

4.2.1 Rueda Cíclica

En esta disciplina (en inglés Round Robin, RR), la pila de prioridades para el uso del procesador se define al inicializar el sistema y de ahí en adelante se arma poniendo a su tope la tarea que sigue, módulo *m*, a la que tuvo la prioridad más alta en la ranura anterior. Si las tareas se numeran de 1 a *m* y la de más alta prioridad en la ranura anterior fue *i*, el ordenamiento para la ranura actual será:

$$[i \bmod m]+1,\ [(i+1) \bmod m]+1,\ \ldots,[(i+m-2) \bmod m]+1,\ i.$$

Existen diferentes variantes de esta disciplina, dependiendo si la tarea que está en el tope de la pila al comienzo de la ranura, requiera o no el procesador. Si el diagramador no asigna la ranura a ninguna tarea si la de mayor prioridad no requiere la utilización del procesador, es decir deja una ranura *ociosa*, la disciplina se denomina Rueda Cíclica Simple (SRR). Si por el contrario, el diagramador asigna el procesador a la tarea que en ese momento tiene la mayor prioridad de todas las que requieren el procesador, la disciplina se denomina Rueda Cíclica Justa (FRR, Fair RR). La disciplina FRR produce una ranura ociosa sólo si no existe ninguna tarea que requiera el procesador al comienzo de dicha ranura.

Esta disciplina es utilizada en sistemas de tiempo compartido, pero su utilización en sistemas de tiempo real no permite elevados factores de utilización del procesador si se desea que el sistema sea diagramable.

4.2.2 Condiciones de diagramabilidad de la disciplina de rueda cíclica

Para el análisis de la diagramabilidad se considera que el sistema está integrado por *m* tareas, expresado: $\mathbf{S}(m)=\{\tau_1,\ \tau_2,\ldots,\ \tau_m\}$. El tiempo de ranura es fijo y es utilizado como unidad de medida para expresar los períodos, tiempos de ejecución y vencimientos de cada una de las tareas del

sistema. El tiempo de ejecución de cada una de las tareas se considera de una ranura. Un sistema de tiempo real, con tareas con tiempos de ejecución unitario, que implementa una disciplina de prioridades por rueda cíclica, es diagramable si y sólo si se satisface:

$$\min_{\forall i | \tau_i \in \mathbf{S}(m)} D_i \geq m .$$

El sistema será diagramable si el menor de los vencimiento de las tareas es mayor que la cantidad de tareas que integran dicho sistema. Esto es porque en el peor caso de carga, la tarea con menor vencimiento tendrá que tolerar la ejecución de las restantes m-1 tareas, antes de poseer al procesador por una ranura para su propia ejecución.

El análisis de sistemas con tareas con tiempos de ejecución arbitrarios se realiza de forma similar.

4.2.2.1 Ventajas del algoritmo de diagramación de rueda cíclica

1. La ventaja más importante de un diagramador con una disciplina de rueda cíclica es su sencillez. La complejidad del algoritmo es muy reducida, lo que permite obtener una implementación confiable y con una relativa baja sobrecarga del procesador.
2. La utilización de esta disciplina en la mayoría de los sistemas de tiempo compartido, ha motivado gran cantidad de investigaciones. Existe una gran bibliografía que cubre diversas características y variaciones de esta disciplina, como también varios Sistemas Operativos que la implementan.

4.2.2.2 Desventajas del algoritmo de diagramación de rueda cíclica

1. La condición que debe cumplir el sistema para que sea diagramable restringe considerablemente su utilización en sistemas de tiempo real. La condición de que todos los vencimientos deben ser mayores que el número de tareas, requiere que las restricciones temporales no sean exigentes o que la complejidad del sistema sea pequeña y pueda implementarse con un reducido número de tareas.

4.2.3 Menor Tiempo al Vencimiento

En esta disciplina, la pila de prioridades se realiza al comienzo de cada ranura, colocando las tareas en orden creciente al tiempo que resta para su vencimiento. La tarea que tiene menor tiempo al vencimiento tiene la mayor prioridad. En esta disciplina el diagramador debe conocer los vencimientos de todas las tareas del sistema, y observar los instantes en que se generan cada una de ellas.

Si la tarea τ_i produce su j-ésima generación en el instante $t_i(j)$, su prioridad, luego de ese instante, será inversamente proporcional a $(t_i(j)+D_i)-t$. El valor $t_i(j)+D_i$ determina el instante en que se producirá el vencimiento de la j-ésima generación de la tarea τ_i. Luego, la expresión $(t_i(j)+D_i)-t$ determina el tiempo que resta, en el instante t, para que la tarea τ_i entre en crisis. La tarea de mayor prioridad en el instante t será la que posea el menor valor de dicha expresión. Si el tiempo al vencimiento para dos o más tareas es el mismo, una regla de desempate debe ser aplicada (p.e. la tarea con menor subíndice tendrá mayor prioridad).

4.2.4 Condiciones de diagramabilidad de la disciplina de Menor Tiempo al Vencimiento

En [19] los autores demuestran formalmente una simple condición necesaria y suficiente de diagramabilidad para sistemas que implementan una disciplina de Menor Tiempo al Vencimiento. En el análisis se considera que el sistema está integrado por m tareas, expresado: $\mathbf{S}(m)=\{\tau_1, \tau_2,..., \tau_m\}$. El tiempo de ranura es fijo y es utilizado como unidad de medida para expresar los períodos, tiempos de ejecución y vencimientos de cada una de las tareas del sistema. Los vencimientos son iguales a los períodos ($D = T$). Los tiempos de ejecución, C, son arbitrarios. Un sistema de tiempo real que implementa una disciplina de prioridades por Menor Tiempo al Vencimiento, es diagramable si y sólo si:

$$\sum_{\forall \tau_i \in \mathbf{S}(m)} \frac{C_i}{T_i} \leq 1.$$

En [19] los autores también demuestran que la disciplina de Menor Tiempo al Vencimiento es óptima en sistemas monoprocesador en el sentido de que, si un sistema no es diagramable por Menor Tiempo al Vencimiento, no lo será por ninguna otra disciplina de prioridades.

4.2.4.1 Ventajas del algoritmo de diagramación por Menor Tiempo al Vencimiento

1. La disciplina de prioridades es óptima en sistemas de tiempo real monoprocesador. Ser óptima significa que si el sistema no es diagramable por la disciplina de Menor Tiempo al Vencimiento, no lo será bajo ninguna disciplina de prioridades. Cabe aclarar, que la condición de óptima es válida para sistemas donde las tareas son independientes y no comparten recursos entre ellas. En sistemas en donde existen recursos compartidos o relaciones de precedencia entre las tareas, y en sistemas multiprocesador, la condición de disciplina de prioridades óptima para sistemas de tiempo real no se mantiene.

4.2.4.2 Desventajas del algoritmo de diagramación por Menor Tiempo al Vencimiento

1. La implementación de un diagramador por Menor Tiempo al Vencimiento es compleja, pues debe considerar los vencimientos y los instantes de generación de cada una de las tareas. En la disciplina por Menor Tiempo al Vencimiento, la estructura del diagramador es más compleja por la cantidad de información de cada tarea que debe considerar.
2. La utilización del procesador por parte del diagramador, produce una sobrecarga en el sistema que disminuye el tiempo que el procesador está disponible para ejecutar las tareas del sistema. Esto se debe a que el diagramador debe utilizar el procesador un tiempo considerable al comienzo de cada ranura para actualizar las prioridades de todas las tareas.

3. El comportamiento del sistema ante una falla por sobrecarga no tiene una degradación parcial, sino que produce una crisis de todas las tareas. Este fenómeno es comúnmente denominado "efecto dominó", puesto que, cuando el sistema está sobrecargado, la primera tarea que pierda su vencimiento podrá causar la pérdida de los vencimientos de todas las tareas que requieran posteriormente el procesador. Este comportamiento es muy indeseable, ya que, en el en "mundo real", pueden ocurrir sobrecargas esporádicas debidas a errores en la determinación de las especificaciones temporales. Sin embargo, estas sobrecargas deberían causar la pérdida de los vencimientos de las tareas menos críticas, y no de la totalidad del sistema.

4.2.4.3 Prioridades Fijas

En esta disciplina (en inglés *Fixed Priority*) la pila de prioridades se arma en el instante inicial de acuerdo a un cierto criterio, para permanecer posteriormente invariante. El algoritmo de diagramación debe seleccionar, al comienzo de cada ranura, la tarea de mayor prioridad en estado de lista.

Si bien la disciplina de menor tiempo al vencimiento necesita que se satisfaga una condición menos rigurosa para que el sistema no entre en crisis (de hecho es óptima en sistemas monoprocesador), la disciplina de prioridades fijas es un estándar *de facto* adoptado por el Departamento de Defensa de los Estados Unidos. Esto se debe a que Prioridades Fijas permite una degradación del sistema más controlada que la disciplina de menor tiempo al vencimientos. En la disciplina de menor tiempo al vencimiento, ante la pérdida ocasional de un vencimiento de alguna de las tareas, se producirá un efecto dominó debido a que el sistema intentará diagramar tareas que ya entraron en crisis. Por el contrario, en Prioridades Fijas las tareas que tienen mayor prioridad seguirán sin perder sus vencimientos aunque tareas de menor prioridad entren en crisis.

Un caso particular de la pila de prioridades se obtiene ordenando a las tareas por sus períodos en forma monotónica creciente, siendo la tarea de

mayor prioridad la que posee el menor período. En este caso se dice que la disciplina de prioridades fijas es de *Períodos Monotónicos Crecientes*. Lo particular de este ordenamiento es que esta disciplina es óptima entre las disciplinas de Prioridades Fijas, en el sentido de que si el sistema no es diagramable utilizando Períodos Monotónicos Crecientes, tampoco lo será con ningún otro ordenamiento de la pila de prioridades [19]. Para obtener esta condición de optimalidad, los autores consideraron tareas periódicas, con vencimientos iguales a sus períodos, independientes y con una diagramador apropiativo. Si los vencimientos son distintos a los períodos, la pila de prioridades debe obtenerse ordenando las tareas por sus vencimientos en forma monotónica creciente.

4.2.4.4 Ventajas del algoritmo de diagramación por Prioridades Fijas.

1. La degradación del sistema ocurre en forma gradual ante sobrecarga del mismo. Al contrario de lo que ocurre con el efecto dominó de la disciplina de menor tiempo al vencimiento, en la disciplina de prioridades fijas, las tareas de mayor prioridad serán las últimas en entrar en crisis cuando se produzca una condición de sobrecarga del sistema. De esta manera, el diseñador puede asignar las mayores prioridades a las tareas cuyos vencimientos producen las situaciones más criticas del sistema.
2. La implementación del algoritmo de diagramación es sencilla. Esto se debe a que el diagramador debe considerar una pila de prioridades que permanece estática durante todo el tiempo de corrida del sistema, sin necesidad de actualizarla al comienzo de cada ranura. Esta sencillez del algoritmo de diagramación permite implementaciones confiables y de baja sobrecarga del procesador.
3. La adopción de la disciplina de Prioridades Fijas por el departamento de Defensa de los Estados Unidos lo impuso como un estándar *de facto*.

4.2.4.5 Desventajas del algoritmo de diagramación por Prioridades Fijas.

1. No es óptimo. Esto significa que existen sistemas diagramables mediante la disciplina de Menor Tiempo al Vencimiento que no lo son utilizando Prioridades Fijas.
2. No existen investigaciones que consideren la disciplina de prioridades fijas con tareas con relaciones de precedencia. Las relaciones de precedencia insertan características particulares a la diagramación por Prioridades Fijas, que no son tratadas con suficiente profundidad en los trabajos de investigación. En esta tesis se desarrollan condiciones de diagramabilidad para sistemas de tiempo real diagramados mediante la disciplina de Prioridades Fijas y con tareas con relaciones de precedencia.

Capítulo 4

En este capítulo se desarrollan los principales resultados obtenidos en las investigaciones de sistemas de tiempo real aplicando la disciplina de Prioridades Fijas. Se detalla el método de las ranuras vacías [32] pues provee una condición de diagramabilidad necesaria y suficiente para sistemas de tiempo real monoprocesador, diagramados mediante la disciplina de Prioridades Fijas. Este método, originalmente desarrollado para redes de comunicación en sistemas de tiempo real, fue demostrado formalmente para el análisis de la diagramabilidad de sistemas multitarea-monoprocesador [31]. Se abarca también la diagramación mediante la disciplina de Prioridades Fijas de sistemas monoprocesadores y multiprocesadores.

1. Diagramabilidad de Sistemas Monoprocesador con Prioridades Fijas

En el capítulo 3 se describieron las disciplinas de diagramación más utilizadas e investigadas en sistemas de tiempo real. Debido a sus características, la disciplina de Prioridades Fijas es un estándar *de facto* por

ser adoptada, primeramente por el Departamento de Defensa de los Estados Unidos y luego por diversas empresas entre las cuales figuran: IBM, Honeywell, McDonnell Douglas, Boeing, General Electric, General Dynamics, Magnavox, Mitre, NASA, Naval Air Warfare Center y Paramax [22].

El primer tratamiento teórico sobre diagramación con Prioridades Fijas en sistemas de tiempo real, fue publicado en 1973, cuando Liu and Layland [19] introdujeron el algoritmo de diagramación por Períodos Monotónicos Crecientes (en inglés *Rate Monotonic Scheduling*) para tareas periódicas e independientes. El término de Períodos Monotónicos Crecientes se debe al método de asignación de prioridades a las tareas, ya que son establecidas en función del período de cada una de las ellas: menor período, mayor prioridad. Con esta simple regla de asignación de prioridades, Liu y Layland demostraron importantes propiedades de los sistemas de tiempo real que utilizan esta disciplina.

La preferencia por adoptar el algoritmo de diagramación por Períodos Monotónicos Crecientes, entre todas las posibilidades de ordenamiento de la pila de prioridades, fue cimentado en el siguiente teorema:

1.1.1 Teorema 4.1 (Liu and Layland [19])

Si existe un ordenamiento por Prioridades Fijas de la lista de prioridades que haga diagramable el sistema, entonces el ordenamiento por Períodos Monotónicos Crecientes hará diagramable al sistema.

El teorema anterior demuestra que la disciplina de Períodos Monotónicos Crecientes es óptima entre las disciplinas de prioridades fijas. Para obtener este resultado, los autores consideraron el sistema de tiempo real formado por un conjunto de tareas periódicas, apropiativas, independientes entre sí y con sus vencimientos iguales a sus respectivos períodos. Estas hipótesis, por ser válidas en un extenso conjunto de implementaciones de sistemas de tiempo real, se conservaron en numerosos

trabajos de investigación posteriores.

Demostrada la optimalidad de la disciplina de Períodos Monotónicos Crecientes entre los diferentes ordenamientos de prioridades fijas, los autores probaron formalmente una cota suficiente para garantizar la diagramabilidad de un sistema de tiempo real que utiliza dicha disciplina.

1.1.2 Teorema 4.2 (Liu and Layland [19])

Un conjunto de *m* tareas independientes y periódicas diagramadas mediante la disciplina de Períodos Monotónicos Crecientes, será diagramable si:

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq m \cdot \left(2^{1/m} - 1\right) \qquad (4.1)$$

C_i / T_i es definido en dicho trabajo como el *factor de utilización* de la tarea τ_i. Este factor representa la fracción de tiempo de procesador utilizado en la ejecución de la tarea τ_i.

Si se aplica la ecuación 4.1, suponiendo que el número de tareas del sistema tiende a infinito ($m \to \infty$), se comprueba que un sistema de tiempo real diagramado por Períodos Monotónicos Crecientes, será diagramable si la suma de los factores de utilización de las tareas es menor que $log_e\ 2 \cong 0{,}693$.

La ventaja principal de esta cota es su muy baja complejidad de cálculo. Lamentablemente, es muy pesimista para la determinación de la diagramabilidad de un sistema de tiempo real. Esto significa que existen muchos sistemas de tiempo real diagramables mediante Períodos Monotónicos Crecientes, que no cumplirán con la cota del teorema 4.2.

Para la obtención de la cota de suficiencia del teorema 4.2, los autores definen *el peor caso de carga* como el instante en el cual las tareas del sistema tendrán su máximo tiempo total de ejecución. La importancia de esta definición reside en el hecho de que si el sistema es diagramable para

el peor caso de carga, entonces lo será para cualquier otro estado arbitrario del sistema. Por lo tanto, es necesario demostrar que el sistema es diagramable en el peor caso de carga, para garantizar que el sistema es diagramable. En [19] se demuestran que el peor caso de carga para la tarea τ_i en un sistema multitarea-monoprocesador, con tareas independientes y periódicas, es la generación simultánea de todas las tareas con mayor prioridad.

Posteriores trabajos de investigación (e.g. [11], [15], [31], [32]) han desarrollado condiciones necesarias y suficientes para determinar la diagramabilidad de sistemas que utilizan la disciplina de Períodos Monotónicos Crecientes. En [32] se demuestra el *método de las ranuras vacías* para determinar en forma necesaria y suficiente la diagramabilidad de un sistema de tiempo real multitarea-monoprocesador.

2. Método de las ranuras vacías.

Este método fue presentado en [32] para resolver el problema de diagramación de Redes Locales en tiempo real. Luego fue extendido en [5] y [31] a la diagramación multitarea apropiativa en monoprocesador aun con granularidad restringida.

El tiempo se considera ranurado y la duración de la ranura es tomada como unidad de tiempo. Las ranuras son denominadas *t* y numeradas 1, 2, ... Las expresiones *instante t* y *comienzo de ranura t* son equivalentes. Un conjunto de *m* tareas τ_1, τ_2, ... , τ_m, asignadas a un procesador, queda completamente especificado como $\mathbf{S}(m)= \{(C_1, T_1, D_1), (C_2, T_2, D_2), ..., (C_m, T_m, D_m)\}$, donde C_i, T_i, D_i representan el máximo tiempo de ejecución, el mínimo tiempo entre arribos y el vencimiento de la tarea τ_i respectivamente.

En [19] se ha demostrado que la disciplina de prioridades por Períodos Monotónicos Crecientes, es la óptima entre las de prioridades fijas en el sentido de que si un sistema es diagramable por alguna disciplina de

prioridades fijas, también lo será por Períodos Monotónicos Crecientes.

En [32] se ha demostrado que la *j*-ésima ranura vacía que deja el sistema de *m* tareas, **S**(*m*), luego del instante de peor estado de carga (arribo simultáneo de todas las tareas en el instante inicial), notada $e_{j(m)}$, es

$$e_{j(m)} = \text{menor}(t) \mid t = j + \sum_{h=1}^{m} C_h \left\lceil \frac{t}{T_h} \right\rceil \tag{4.2}$$

En el mismo trabajo, se demuestra que, un sistema **S**(*m*) es diagramable bajo Períodos Monotónicos Crecientes si y sólo si para todo *i* =1, 2, 3, .. *m*, se cumple $D_i \geq e_{C_i(i-1)}$.

Trabajos posteriores a [31] modificaron el método de las ranuras vacías para determinar la diagramabilidad de sistemas de tiempo real con un reducido número de prioridades, con diagramadores que implementan la discipliana de prioridades pero luego de *k*-1 ranuras (denominados *k*-diagramables), y con inversiones de prioridad. La posibilidad de adaptar este método para analizar la diagramabilidad de sistemas con diferentes características reside en la propiedad de que, el peor caso de carga enunciado por Liu y Layland en [19], se conserva para todos ellos. Sin embargo, cuando existen relaciones de precedencia entre las tareas que integran el sistema, no es válido considerar como peor caso de carga el demostrado por Liu y Layland. Esta característica de las relaciones de precedencia hace que dichos sistemas sean tratados con consideraciones especiales.

En [7] se demuestran condiciones para determinar la diagramabilidad de un sistema con tareas con relaciones de precedencia que utiliza la disciplina de Menor Tiempo al Vencimiento. Sin embargo, no existen trabajos que demuestren condiciones de diagramabilidad para sistemas monoprocesador, con tareas con relaciones de precedencia generales, que utilicen la disciplina de Prioridades Fijas. A continuación se especificará

formalmente las relaciones de precedencia.

3. Relaciones de Precedencia

Es usual que en sistemas de tiempo real resulte necesario que las tareas no sean independientes y cooperen entre sí para lograr un determinado objetivo. Cuando una tarea τ_j necesita total o parcialmente los resultados producidos por otra tarea τ_i para poder ser ejecutada, se establece una relación de precedencia, notada $\tau_i \prec \tau_j$ (τ_i precede a τ_j), la cual establece un orden parcial sobre las tareas. Si $\tau_i \prec \tau_j$ y no existe una tarea τ_l tal que $\tau_i \prec \tau_l$ y $\tau_l \prec \tau_j$, entonces τ_i y τ_j son denominadas predecesora y sucesora respectivamente. Este modelo de computación puede ser asociado a un grafo **G** [41]. Cada nodo del mismo simboliza una tarea y existirá un arco dirigido desde el nodo que representa τ_i al nodo que representa τ_j si y sólo si ellas son predecesora y sucesora respectivamente. **G** = {Γ, P}, donde Γ es el conjunto de tareas que forman el trabajo y P es el conjunto de sus relaciones de precedencia.

Una relación de precedencia agrega complejidad a la diagramación. En [14] se ha resuelto en forma óptima, con complejidad $\mathbf{O}(n^2)$, un problema simple de un conjunto de *n* tareas no apropiativas, idéntico tiempo de arribo y con condiciones de precedencia. Este método, lamentablemente, no es suficiente para la mayoría de los problemas de interés en sistemas de tiempo real, en los cuales las tareas no tienen un instante conocido de arribo. En [16] se ha demostrado que el problema de diagramación de tareas con diferentes tiempos de arribo y con relación de precedencia es NP-duro. Los mejores resultados sobre precedencia fueron obtenidos a partir del trabajo sobre subclases de la relación de precedencia, alcanzándose algoritmos polinómicos para relaciones *intrees* (un solo sucesor) y *outtrees* (un solo predecesor). Estos casos son incluidos en un grupo más interesante denominado serie-paralelo que es definido en forma

recursiva y sobre el cual existen soluciones eficientes, aunque, lamentablemente, los problemas no son comunes en la práctica. Si el sistema es apropiativo, se reduce la complejidad del diagramador y es solucionable en $\mathbf{O}(n^2)$ aplicando el algoritmo de Baker [26].

En esta tesis se entenderá por *trabajo*, notado **J**, a todo el conjunto de tareas que tienen relaciones de precedencia entre ellas. Los trabajos son independiente entre ellos, esto significa que no existe relaciones de precedencia entre tareas pertenecientes a distintos trabajos.

En el capítulo 5 se demostrará que el peor caso de carga hallado por Liu y Layland no se conserva cuando existen relaciones de precedencia entre las tareas. También se mostrará que la disciplina por Períodos Monotónicos Crecientes no es óptima entre las disciplinas de Prioridades Fijas cuando las tareas tienen relaciones de precedencia.

4. Resultados Monoprocesador vs. Resultados Multiprocesador.

Diversos algoritmos y resultados teóricos han sido desarrollados para el estudio de la diagramabilidad en sistemas de tiempo real duro multitarea-monoprocesador. La mayoría de estos resultados se basan en la utilización de las disciplinas de Menor Tiempo al Vencimiento y Períodos Monotónicos Crecientes. Se han establecidos condiciones necesarias y suficientes para determinar la diagramabilidad de sistemas multitarea-monoprocesador con relaciones de precedencia, recursos compartidos, tareas apropiativas y no apropiativas, inversiones de prioridad, etc. Sin embargo, la mayoría de los resultados en sistemas multitarea-multiprocesador determinan que su diagramación posee complejidad NP-duro [40].

4.1 Optimalidad monoprocesador vs. multiprocesador.

Decimos que un algoritmo de diagramación es *óptimo*, si cuando se

aplica a un sistema de tiempo real y éste entra en crisis, entonces también lo hará con cualquier otro algoritmo de diagramación.

Uno de los resultados más importantes en la teoría de diagramación es la determinación de que la disciplina de Menor Tiempo al Vencimiento es óptima entre los sistemas de tiempo real duro multitarea-monoprocesador.

4.1.1 Teorema 4.3 (Liu and Layland [19])

Dado un conjunto de m tareas independientes, $\mathbf{S}(m)$, éste será diagramable por la disciplina de Menor Tiempo al Vencimiento si y sólo si:

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq 1$$

El teorema 4.3 asegura que la disciplina de Menor Tiempo al Vencimiento es óptima en el sentido de que si el conjunto de tareas no puede ser diagramado por dicha disciplina, tampoco lo será bajo cualquier otro algoritmo de diagramación. Sin embargo, el siguiente teorema afirma que la optimalidad de la disciplina de Menor Tiempo al Vencimiento no se conserva en sistemas multiprocesador.

4.1.2 Teorema 4.4 (Mok [21])

La disciplina de Menor Tiempo al Vencimiento no es óptima para sistemas multitarea-multiprocesador.

Ejemplo:

Consideremos un sistema de tres tareas, $\mathbf{S}(m)=\{\tau_1, \tau_2, \tau_3\}$, que deben ejecutarse en dos procesadores. El vencimiento y tiempo máximo de ejecución para cada tarea está dado por la siguiente tabla:

	C	D
τ_1	1	1
τ_2	1	2
τ_3	3	4

Luego de un arribo simultáneo de τ_1, τ_2 y τ_3, un diagramador que utiliza la disciplina de Menor Tiempo al Vencimiento, ilustrado en la figura

4.1, ejecutará τ_1 en el procesador 1, τ_2 en el procesador 2 y entonces τ_3 perderá su vencimiento.

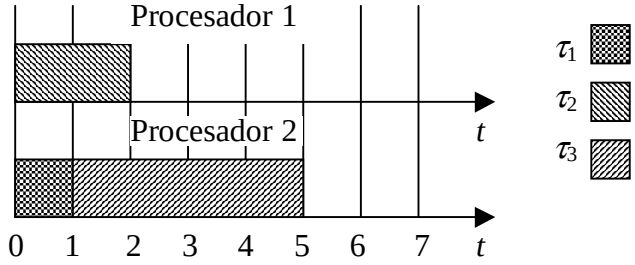


Figura 4.1

Sin embargo, si se ejecuta τ_3 en el procesador 1 y τ_2 y τ_3 en el procesador 2, el sistema será diagramable puesto que ninguna tarea perderá su vencimiento. Este caso de ejecución es representado en el diagrama temporal de la figura 4.2.

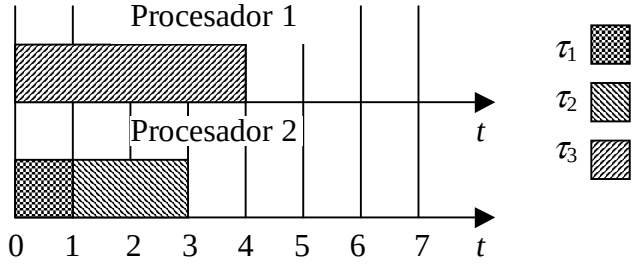


Figura 4.2

El ejemplo anterior prueba que la disciplina de Menor Tiempo al Vencimiento no es óptima para la diagramación de sistemas multitarea-multiprocesador. El paso siguiente es saber si existe algún algoritmo de diagramación óptimo para sistemas multiprocesador. El teorema 4.5 demuestra que no existe tal algoritmo, dejando de esta manera una atmósfera pesimista en la diagramación multiprocesador.

4.1.3 Teorema 4.5 (Mok [21])

Para un sistema con dos o más procesadores, ningún algoritmo basado en vencimientos puede ser óptimo sin un completo conocimiento *a priori* de los vencimientos, máximos tiempos de ejecución e instantes de arribos.

El teorema 4.5 implica que, debido a que los algoritmos deben tener conocimiento de los futuros arribos de las tareas, ningún algoritmo

dinámico puede ser óptimo en sistemas de tiempo real multitarea-multiprocesador. Esto implica que es imposible desarrollar una disciplina de diagramación dinámica óptima general. Sin embargo, pueden desarrollarse algoritmos de diagramación óptimos para sistemas con condiciones particulares.

En los sistemas multiprocesador existe una anomalía, denominada anomalía de Richard, que el diseñador debe considerar en el momento de proyectar su sistema.

4.1.4 Anomalía de Richard [9]

Dado un conjunto de tareas óptimamente diagramado con alguna disciplina de prioridades sobre un conjunto fijo de procesadores, tiempos de ejecución fijos y relaciones de precedencia, entonces el sistema puede dejar de ser diagramable si: se modifica la pila de prioridades, se incrementa el número de procesadores, se reduce el tiempo de ejecución de las tareas o se alteran las relaciones de precedencia.

En un principio, es poco intuitivo pensar que agregando procesadores al sistema, aumentando la velocidad de procesamiento de los mismos o disminuyendo las relaciones de precedencia, el sistema pueda hacerse no diagramable. Esto demuestra la particular naturaleza de los sistemas multitarea-multiprocesador en tiempo real duro.

5. Disciplina de Períodos Monotónicos Crecientes en Sistemas Multiprocesador: Diagramación Local y Global.

La diagramación en un sistema multitarea-multiprocesador puede ser realizada por un diagramador con alcance local o global. En un diagramador global la disciplina de diagramación es aplicada al conjunto total de tareas sobre todos los procesadores. Si es de alcance local, existe

un diagramador autónomo en cada procesador que actúa sobre las tareas que le fueron previamente asignadas.

Los sistemas multiprocesador con diagramador global constan de un procesador principal que ejecuta el diagramador, y un conjunto de procesadores con funciones no específicas. Los procesadores pueden ser conectados en diversas topologías: anillo, cubo, hipercubo, barra. Existe un procesador, denominado *procesador diagramador* (figura 4.3), encargado de aplicar la disciplina de diagramación sobre la totalidad de las tareas del sistema, y determinar, en forma dinámica, el procesador en que se ejecutarán. La principal desventaja de los sistemas con diagramación global es la gran degradación de performance que produce una falla en el procesador diagramador o en la red de comunicaciones. Si consideramos que una de las ventajas que motiva la utilización de la disciplina de Prioridades Fijas es su controlada degradación ante fallas, esta ventaja se puede ver anulada si se utiliza en un diagramador global. Otra desventaja es la complejidad que provoca en el algoritmo de diagramación-asignación, la utilización de un diagramador global en sistemas distribuidos con procesadores heterogéneos. En estos casos, el diagramador debe considerar, en el momento de asignación, si el procesador tiene todos los recursos necesarios para que la tarea se ejecute completamente, o si su código puede ser ejecutado en dicho procesador.

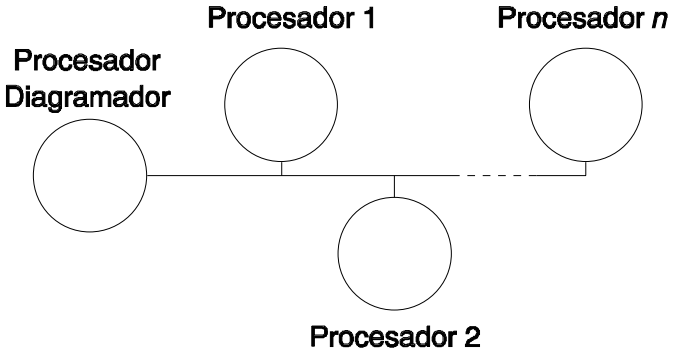


Figura 4.3

En los sistemas con diagramación local las tareas son asignadas a cada procesador de manera fija y específica. El diagramador de cada procesador

tiene conocimiento en todo momento de las tareas que le fueron asignadas para diagramar. La asignación de las tareas en cada procesador es realizada en tiempo de diseño y permanece constante en tiempo de ejecución. La degradación del sistema es más gradual que en la diagramación global ante la falla de algún procesador o de la red de comunicaciones. Por este motivo, la mayor confiabilidad que poseen los sistemas multiprocesadores con diagramación local los hacen más aptos para sistemas de tiempo real duro.

El siguiente ejemplo compara la utilización de un algoritmo de diagramación global y uno local. La disciplina de diagramación a utilizar en ambos ejemplos será Períodos Monotónicos Crecientes. Consideremos tres tareas periódicas τ_1, τ_2 y τ_3 que deben ser ejecutadas en dos procesadores. τ_1 y τ_2 tienen intervalos entre generaciones de 50 ranuras y máximos tiempo de ejecución por período de 25 ranuras; τ_3 tiene 100 ranuras entre arribos y su máxima ejecución demanda 80 ranuras. Si la disciplina por Períodos Monotónicos Crecientes es aplicada, las tareas τ_1 y τ_2 tendrán las más altas prioridades, mientras que la tarea τ_3 tendrá la menor. Ante un arribo simultáneo, considerando el caso de una diagramación global, las tareas τ_1 y τ_2 se ejecutarán en forma paralela en cada uno de los procesadores. Cuando τ_1 y τ_2 finalicen su ejecución, dejarán disponibles 75 ranuras para la ejecución de τ_3. El hecho de que τ_3 tenga disponible dos procesadores para su ejecución es irrelevante pues uno permanecerá ocioso. Como resultado de aplicar un algoritmo de Períodos Monotónicos Crecientes con diagramador global, τ_3 entrará en crisis aunque el factor de utilización promedio sea solamente del 65%. Sin embargo, la utilización de períodos monotónicos crecientes con diagramación local y asignando τ_1 y τ_2 a un procesador y τ_3 al otro, permite que todas las tareas cumplan con sus vencimientos.

Capítulo 5

En este capítulo se muestran algunos resultados obtenidos a partir del análisis de sistemas de tiempo real diagramados por un diagramador por Prioridades Fijas. Estos resultados muestran que la posibilidad de diagramar un sistema aumenta si se establecen relaciones de precedencia entre las tareas. Luego, una adecuada partición de tareas en subtareas con relaciones de precedencia de menor duración, permitirá, eventualmente, diagramar un sistema que en un principio era no-diagramable. Finalmente, se demuestran formalmente condiciones suficientes para garantizar la diagramabilidad de un sistema monoprocesador con tareas con relaciones de precedencia.

1. Introducción

Desde el trabajo liminar de Liu y Layland [19], diversos métodos han sido propuestos para resolver el problema de diagramar un conjunto de tareas en sistemas multitarea-monoprocesador, con eficacia creciente tanto desde; el punto de vista de la utilización del procesador [11, 15, 17, 31], como el de compartir otros recursos [35, 45] operando bajo distintas

disciplinas de prioridades. De ellas, la más importante, por ser un estándar *de facto* impuesto por el Departamento de Defensa de EEUU, es la denominada de Prioridades Fijas.

Un problema adicional, no tratado en [19], es que entre pares de tareas, pueden existir relaciones de precedencia. Se entiende por ello que, para comenzar su ejecución, una de ellas (*sucesora*) necesita datos producidos por otra (*predecesora*). La llegada de dichos datos desde todas las predecesoras a la tarea sucesora define el instante de generación de la misma. Por otro lado, si las tareas no poseen relaciones de precedencia entre ellas, se dice que las tareas son independientes.

En la siguiente sección se mostrará que, cuando existen relaciones de precedencia entre las tareas, aumentan las posibilidades de diagramar el sistema para igual factor de utilización del procesador.

El método de las ranuras vacías será utilizado para la determinación de la diagramabilidad.

2. Diagramación de tareas con relaciones de precedencia en sistemas monoprocesador.

En un sistema multitarea-monoprocesador con tareas independientes, el peor caso de carga es la generación simultánea de todas las tareas [19]. Al existir relaciones de precedencia, no todas las tareas podrán generarse simultáneamente. Es lógico que al necesitar las tareas sucesoras la completa ejecución de sus respectivas predecesoras, no podrá darse el arribo simultáneo de la sucesora y sus correspondientes predecesoras. Esta consideración permite intuir que en sistemas donde las tareas poseen una adecuada relación de precedencia, el peor caso de carga puede ser más fácilmente diagramable que en sistemas con tareas independientes. Es importante notar que cuando existen relaciones de precedencia entre las tareas, no es válido afirmar que el peor caso de carga es la generación simultánea. De esta manera, en la mayoría de los sistema no diagramables

por Períodos Monotónicos Crecientes, existe la posibilidad de convertir una tarea independiente en un conjunto de tareas con relaciones de precedencia, que permitiera hacer diagramable al sistema.

A continuación se ilustrará que el aprovechamiento de las relaciones de precedencia, depende del ordenamiento que poseen las tareas en la pila de prioridades.

3. Alternativas en la asignación de prioridades.

Consideraremos un sistema: $\mathbf{S}(5)=\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$, en donde $\tau_4 \prec \tau_5$, mientras que τ_1, τ_2 y τ_3 son tareas independientes. La figura 5.1 ilustra las relaciones de precedencia de las tareas que integran el sistema. A continuación se consideran las distintas alternativas que pueden darse en la asignación de prioridades a las tareas.

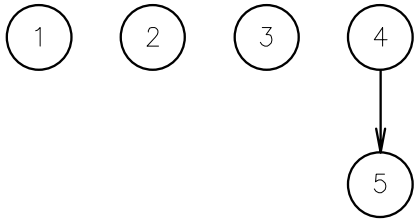


Figura 5.1

3.1 Alternativa 1

Consideremos el sistema con la siguiente pila de prioridades $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$ siendo τ_1 la tarea de mayor prioridad. Notaremos que esta pila de prioridades cumple con la disciplina de prioridades de períodos monotónicos crecientes. Para que un sistema de tiempo real que utiliza la disciplina de Prioridades Fijas resulte diagramable debe cumplirse [31] que:

$$\forall i, \quad T_i \geq D_i \geq \min t | t = C_i + \sum_{h=1}^{i-1} C_h \cdot \left\lceil \frac{t}{T_h} \right\rceil = e_{C_i(\mathbf{S}(i-1))} \qquad (5.1)$$

Aplicando el método de las ranuras vacías, un sistema con inversión de

prioridades será diagramable si:

$$\forall i, \qquad T_i \geq e_{(C_i+k)(\mathbf{S}(i-1))} \tag{5.2}$$

donde k es el máximo número de inversiones a la que puede verse sometida τ_i.

Luego, para que τ_3 y el trabajo formado por las tareas τ_4 y τ_5 resulten diagramables, debe satisfacerse:

$$\begin{aligned} D_3 &\geq \min t \mid t = C_3 + C_1 \cdot \left\lceil \frac{t}{T_1} \right\rceil + C_2 \cdot \left\lceil \frac{t}{T_2} \right\rceil \\ D_5 &\geq \min t \mid t = C_5 + C_1 \cdot \left\lceil \frac{t}{T_1} \right\rceil + C_2 \cdot \left\lceil \frac{t}{T_2} \right\rceil + C_3 \cdot \left\lceil \frac{t}{T_3} \right\rceil + C_4 \cdot \left\lceil \frac{t}{T_4} \right\rceil = \\ &= C_4 + C_5 + C_1 \cdot \left\lceil \frac{t}{T_1} \right\rceil + C_2 \cdot \left\lceil \frac{t}{T_2} \right\rceil + C_3 \cdot \left\lceil \frac{t}{T_3} \right\rceil \end{aligned} \tag{5.3}$$

Nótese que en la expresión anterior $\lceil t/T_4 \rceil = 1$ ya que las tareas τ_4 y τ_5, al estar vinculadas por una relación de precedencia, tienen el mismo período y, por ser contiguas en la pila de prioridades, pueden considerarse como una tarea cuyo tiempo de ejecución es $C_4 + C_5$. Un razonamiento análogo es aplicado en [31] para extender el método de las ranuras vacías a sistemas con tareas con tiempos de ejecución no unitarios.

3.2 Alternativa 2

Consideremos el sistema con la siguiente pila de prioridades $\{\tau_1, \tau_2, \tau_3, \tau_5, \tau_4\}$, siendo τ_1 la tarea de mayor prioridad. Este caso de ordenamiento provoca un comportamiento del sistema idéntico al de la alternativa anterior. La tarea τ_3 debe satisfacer la misma inecuación que en la alternativa 1, mientras que el trabajo compuesto por las tareas τ_4 y τ_5 será diagramable si la (C_4+C_5)-*ésima* ranura libre que deja el sistema $\mathbf{S}(3)$, $e_{(C_4+C_5)(\mathbf{S}(3))}$, es menor que el vencimiento de dicho trabajo.

La razón de esta consideración se basa en que la subtarea τ_4 podrá ser apropiada por las tareas τ_1, τ_2 y τ_3, pero no por τ_5 pues, por ser su

sucesora, necesita los datos producidos por la ejecución completa de τ_4. Luego, la subtarea τ_5 deberá soportar la carga producida por las tareas τ_1, τ_2 y τ_3. De esta manera, la tarea integrada por las subtareas τ_4 y τ_5 debe soportar la carga producida por las tareas τ_1, τ_2 y τ_3, tal como si fuera una tarea independiente con un tiempo de ejecución igual a la suma de C_4 y C_5.

3.3 Alternativa 3

Consideremos el sistema con la siguiente pila de prioridades $\{\tau_1, \tau_2, \tau_4, \tau_3, \tau_5\}$, siendo τ_1 la tarea de mayor prioridad. Para que el sistema sea diagramable deben satisfacerse las siguientes inecuaciones:

$$D_3 \geq \min t \mid t = C_3 + C_1 \cdot \left\lceil \frac{t}{T_1} \right\rceil + C_2 \cdot \left\lceil \frac{t}{T_2} \right\rceil + C_4 \cdot \left\lceil \frac{t}{T_4} \right\rceil$$

$$D_5 \geq \min t \mid t = C_4 + C_5 + C_1 \cdot \left\lceil \frac{t}{T_1} \right\rceil + C_2 \cdot \left\lceil \frac{t}{T_2} \right\rceil + C_3 \cdot \left\lceil \frac{t}{T_3} \right\rceil$$

El vencimiento de la tarea τ_3 debe ser mayor que en la alternativa 1 debido a que, en el peor caso de generación, debe tolerar la ejecución de τ_4 por ser de mayor prioridad. Sin embargo, para la tarea τ_5 el máximo tiempo de ejecución permanece invariante debido a que sólo se modifica el sumando de la tarea τ_4. Como t debe ser menor o igual a T_4 para que el trabajo compuesto por las tareas τ_4 y τ_5 sea diagramable, el factor que tiene C_4, por la ecuación de las ranuras vacías, es igual a uno. De esta manera podemos afirmar que si la tarea predecesora posee mayor prioridad que la sucesora, ambas tareas pueden considerarse, para el estudio de la diagramabilidad de la sucesora, como una tarea con tiempo de ejecución igual a la suma de los tiempo de ejecución de ambas tareas, y con prioridad igual a la prioridad de la sucesora.

3.4 Alternativa 4

Consideremos el sistema con la siguiente pila de prioridades $\{\tau_1, \tau_2, \tau_5, \tau_3, \tau_4\}$, siendo τ_1 la tarea de mayor prioridad. Para que el sistema sea diagramable, deben satisfacerse las siguientes inecuaciones:

$$D_3 \geq \min t \mid t = C_3 + C_1 \cdot \left\lceil \frac{t}{T_1} \right\rceil + C_2 \cdot \left\lceil \frac{t}{T_2} \right\rceil + C_5 \cdot \left\lceil \frac{t}{T_5} \right\rceil$$

τ_5 sólo podrá ejecutarse luego de τ_4 y ambas deberán soportar la carga producida por τ_3 hasta la completa ejecución de τ_4. Esta finalizará su ejecución en $e_{C_4(\{\tau_1,\tau_2,\tau_3\})}$, ya que τ_5 no puede interferir su ejecución. Luego, para τ_5 las ranuras ocupadas por la tarea τ_3 serán $k = \left(C_3 \cdot \left\lceil e_{C_4(\{\tau_1,\tau_2,\tau_3,\tau_4\})} / T_3 \right\rceil\right)$, por lo que

$$D_5 \geq \min t \mid t = \left(C_3 \cdot \left\lceil \frac{e_{C_4(\{\tau_1,\tau_2,\tau_3\})}}{T_3} \right\rceil \right) + C_4 + C_5 + C_1 \cdot \left\lceil \frac{t}{T_1} \right\rceil + C_2 \cdot \left\lceil \frac{t}{T_2} \right\rceil \qquad (5.4)$$

Si tomamos t_1 y t_4 como las soluciones de las expresiones de las alternativas 1 y 4 respectivamente, vemos que t_1 es la primera ranura libre que deja el sistema $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$ y $e_{C_4(\{\tau_1,\tau_2,\tau_3\})}$ es la primera ranura libre que deja el sistema $\{\tau_1, \tau_2, \tau_3\}$ con lo que $t_1 > e_{C_4(\{\tau_1,\tau_2,\tau_3\})}$, luego

$$C_3 \left\lceil \frac{t_1}{T_3} \right\rceil \geq C_3 \left\lceil \frac{e_{C_4(\{\tau_1,\tau_2,\tau_3\})}}{T_3} \right\rceil$$

De esta manera, la inecuación que debe satisfacer la tarea τ_5 en la alternativa 4 es menos rigurosa que la de las alternativas anteriores.

Del estudio de los diferentes ordenamientos que pueden tener las tareas con relación de precedencia, se hace notoria la conveniencia de otorgar a las tareas sucesoras mayor prioridad que a las predecesoras.

4. Conveniencia de las relaciones de precedencias

Una característica interesante, que puede ser aprovechada si se utilizan los resultados de la sección anterior, muestra la conveniencia de particionar las tareas en subtareas con relaciones de precedencia puesto que, de esta manera, el sistema presenta más posibilidades de resultar diagramable. Por ejemplo: es preferible tener dos tareas, predecesora y sucesora, con tiempos de ejecución unitarios, en lugar de una tarea con tiempo de ejecución igual a 2. Esto se debe a que las relaciones de precedencia tienen un mecanismo implícito de diagramación que, utilizado correctamente, puede mejorar la condiciones de diagramabilidad del sistema.

La conveniencia de utilizar relaciones de precedencia en un sistema diagramado por Prioridades Fijas, puede ser descripta estudiando el comportamiento de un diagramador por Menor Tiempo al Vencimiento. La disciplina de Menor Tiempo al Vencimiento es óptima para sistemas monoprocesador con tareas independientes. En esta disciplina la prioridad de cada una de las tareas es determinada por el diagramador al comienzo de la ranura y es inversamente proporcional al tiempo que le resta a dicha tarea para entrar en crisis. De esta manera, cuando más cerca esté de alcanzar su vencimiento, mayor será su prioridad.

Por otro lado, en la disciplina de Prioridades Fijas las tareas poseen una prioridad que permanece invariante durante todo el tiempo de corrida del sistema. Las prioridades son asignadas a cada una de las tareas en tiempo de diseño, y es el único parámetro que el diagramador evalúa para asignar el procesador en cada ranura. Esta forma de diagramación, aunque adoptada como estándar en sistemas de tiempo real duro, no es óptima. Esto significa que existirán sistemas diagramables por la disciplina de Menor Tiempo al Vencimiento que no lo serán si se utiliza una disciplina de Prioridades Fijas.

La inserción de relaciones de precedencia transforma una tarea independiente en un conjunto de tareas, denominado trabajo, que,

representado en un grafo, conforma una cadena (todas las tareas que integran el trabajo tienen una sola predecesora y sucesora, excepto la raíz y la hoja). El período y el vencimiento del trabajo son iguales al período y vencimiento de la tarea original y su tiempo de ejecución será la sumatoria de los tiempos de ejecución de las tareas que integran el trabajo que es igual al tiempo de ejecución de la tarea original. Para poder ejecutarse, las tareas sucesoras necesitan que su correspondiente predecesora complete su ejecución. Esto hace que las tareas sucesoras requieran el procesador en ranuras más cercanas al vencimiento del trabajo que sus correspondientes predecesoras. Utilizando, para asignar las prioridades, la filosofía de la disciplina de Menor Tiempo al Vencimiento (mayor prioridad cuando más cerca del vencimiento), a las tareas sucesoras le corresponderá mayor prioridad que sus respectivas predecesoras. Debido a esto, la prioridad del trabajo irá aumentando, aunque se utilice la disciplina de Prioridades Fijas, a medida que se acerque a su vencimiento. De esta manera, al modificar la prioridad del trabajo, podrán hacerse diagramables sistemas cuya única forma de lograrlo, era modificando, en tiempo de corrida, la prioridad de la tarea independiente original mediante la disciplina de Menor Tiempo al Vencimiento.

Para ilustrar el efecto que presentan las relaciones de precedencia en la diagramación, consideremos el siguiente ejemplo:

Sea el sistema $\mathbf{S}(3)=\{\tau_1, \tau_2, \tau_3\}$, formado por tareas independientes con los siguientes parámetros:

	C_i	T_i
τ_1	3	6
τ_2	3	8
τ_3	1	8

Tabla 5.**I**

La figura 5.2 muestra la evolución del sistema en el peor caso de carga. En este tipo de diagrama, en el eje de las abcisas se representa el tiempo y la tarea que utiliza el procesador en cada ranura. La generación de cada

tarea es representada por una flecha descendente al comienzo de la ranura correspondiente. Puesto que es un sistema que cumple con las condiciones de Liu y Layland, el peor caso de carga es la generación simultánea seguida de requerimientos con el mínimo período. Como puede observarse, la tarea τ_3 entra en crisis en la ranura 8 por no poder ejecutarse antes de su vencimiento.

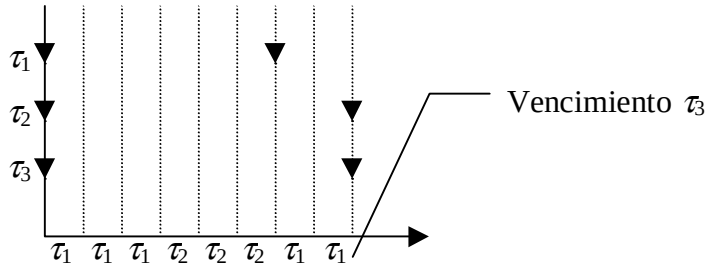


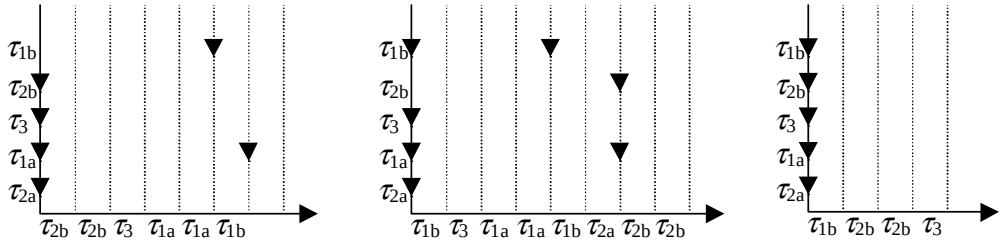
Figura 5.2

Estudiemos el caso en que las tareas τ_1 y τ_2 están compuestas, cada una, por un par de tareas sucesora y predecesora. τ_1 se descompondrá en τ_{1a} y τ_{1b}, predecesora y sucesora respectivamente. τ_2 se descompondrá en τ_{2a} y τ_{2b}, predecesora y sucesora respectivamente. El sistema queda conformado por $\mathbf{S}(5)=\{\tau_{1a}, \tau_{1b}, \tau_{2a}, \tau_{2b}, \tau_3\}$, con los parámetros mostrados en la Tabla **5.II**:

	C_i	T_i
τ_{1a}	2	6
τ_{1b}	1	6
τ_{2a}	1	8
τ_{2b}	2	8
τ_3	1	8

Tabla 5.**II**

Consideremos que el diagramador del sistema tiene la siguiente pila de prioridades: τ_{1b}, τ_{2b}, τ_3, τ_{1a}, τ_{2a}, siendo τ_{1b} la tarea de mayor prioridad. Las figuras 5.3.a, 5.3.b y 5.3.c muestran los peores casos de generación para los trabajos τ_1, τ_2 y τ_3 respectivamente. Las tareas sucesoras son generadas al término de ejecución de su correspondiente predecesora.



5.3.a 5.3.b 5.3.c

Figura 5.3

En un sistema de tiempo real, en donde existen relaciones de precedencia entre las tareas que lo integran, el peor caso de carga enunciado por Liu y Layland no es aplicable. Por este motivo, se considera en este ejemplo que en el instante inicial, cuando genera la tarea raíz del trabajo, existen requerimientos pendientes de todas las tareas que integran dicho trabajo. Una excepción se realiza para el trabajo cuya diagramabilidad se está analizando, puesto que no puede haber requerimientos de la sucesora sin la ejecución completa de la/s respectiva/s predecesora/s si dicho trabajo es diagramable.

Puede observarse que ninguna tarea alcanza su vencimiento. También puede notarse que las tareas no están ordenadas por períodos monotónicos crecientes, y que el sistema es diagramable mediante Prioridades Fijas debido a la diagramación implícita que tienen las relaciones de precedencia.

Nuevamente es posible observar que, para obtener condiciones de diagramación menos rigurosas, es conveniente asignar mayores prioridades a las tareas sucesoras que a sus respectivas predecesoras. De esta manera es aprovechada la diagramación interna que presentan las relaciones de precedencia entre las tareas. Por supuesto, existe una relación de compromiso puesto que puede decaer la eficiencia del diagramador ante un aumento del número de tareas. Por otro lado es posible, en términos prácticos, que la partición de una tarea de tiempo de ejecución C_i en n subtareas conduzca a un conjunto de tareas cuyo tiempo de ejecución total sea mayor que el de la tarea original ($\sum_{h=1}^{n} C_{i,h} \geq C_i$). Esto se debe a que

resulta necesario implementar mecanismos de comunicación entre las tareas que previamente no eran necesarios. A pesar de la complejidad del problema, su tratamiento es importante ya que, como se ha ejemplificado, existen problemas intratables para un diagramador de Prioridades Fijas estándar que, con un adecuado tratamiento de las relaciones de precedencia entre subtareas, puede tornarse factible.

5. Análisis del peor caso de carga para Sistemas Multitarea-Monoprocesador con tareas con relaciones de precedencia.

Los métodos de análisis de diagramabilidad desarrollados para sistemas multitarea-monoprocesador [1, 11, 15, 31] se basan en la hipótesis de que no existen relaciones entre las tareas. En [7] se estudia la diagramabilidad de tareas con precedencia en sistemas monoprocesador, pero aplicando la disciplina de Menor Tiempo al Vencimiento.

En la sección anterior se comprobó que la disciplina de diagramación por Períodos Monotónicos Crecientes no es la disciplina de Prioridades Fijas óptima cuando entre las tareas existen relaciones de precedencia. También se puede demostrar fácilmente que la generación simultánea no es el peor caso de carga en sistemas monoprocesador con relaciones de precedencia.

Cuando existen relaciones de precedencia entre las tareas, es necesario la ejecución completa de la tarea predecesora para generar su/s correspondiente/s sucesora/s. Si bien una tarea puede tener generación periódica por ser tarea raíz, y generarse de esta forma con el período del trabajo al que pertenece, la generación de su sucesora puede producirse a un intervalo aun menor. Esto se debe a que diferencias en la diagramación de la predecesora en dos períodos consecutivos provoca la generación de la sucesora con un intervalo distinto que su predecesora. Eventualmente la sucesora puede generarse con un intervalo menor que el período del trabajo

al que pertenece. Por otra parte, la generación simultánea de todas las tareas de mayor prioridad no será posible si existen relaciones de precedencia entre un subconjunto de ellas (por ejemplo, la predecesora y su sucesora no podrán generarse nunca simultáneamente). Teniendo en cuenta estas particularidades, se puede deducir que no es posible aplicar el análisis del peor caso de carga desarrollado en [19] a sistemas monoprocesador con tareas con relaciones de precedencia. En esta sección se propondrán condiciones de suficiencia para garantizar la diagramabilidad de un sistema monoprocesador con relaciones de precedencia entre sus tareas. Se considerará que los vencimientos de los trabajos son menores o a lo sumo iguales que sus correspondientes períodos ($D \leq T$). En este capítulo se considerarán que las relaciones de precedencia de las subtareas conforman grafos en forma de cadenas (todas las subtareas poseen sólo una predecesora y sucesora, excepto la raíz y la hoja).

5.1 Definición 5.1

Se simbolizará con $P(\tau_i)$ a la prioridad de la tarea τ_i. La proposición $P(\tau_i) > P(\tau_j)$ será verdadera si la tarea τ_i tienen asignada una prioridad mayor que la tarea τ_j.

5.2 Lema 5.1

Una tarea τ_i, raíz de un trabajo con período T, puede tener un máximo de $\left\lceil t/T \right\rceil$ generaciones en un intervalo de t ranuras.

Demostración:

Si la generación del trabajo es periódica, la generación de las tareas raíces de dicho trabajo también lo serán. La generación de un trabajo implica la generación de las tareas raíces del mismo. El máximo número de generaciones en un intervalo t se dará cuando el tiempo entre generaciones sea su período. Con esta cota, el número máximo de generaciones en un

intervalo t estará dado por $\lceil t/T \rceil$.

5.3 Lema 5.2

Una tarea τ_j, sucesora de una tarea τ_i, y perteneciente a un trabajo con período T, puede tener un máximo de $\lceil t/T \rceil$+1 generaciones en un intervalo de t ranuras.

Demostración:

El máximo número de generaciones que pueden tener las tareas raíces de un trabajo, ocurren con un intervalo mínimo entre ellas igual al mínimo tiempo entre generaciones del trabajo (T). La ejecución completa de la predecesora produce la generación de su correspondiente sucesora. Todas las tareas del trabajo deben ejecutarse antes de la próxima generación del trabajo al que pertenecen para ser diagramable. Por este motivo no pueden existir dos generaciones de una tarea sucesora, entre dos generaciones consecutivas de su correspondiente predecesora. Sin embargo, diferencias en los tiempos de ejecución de la tarea predecesora, provoca generaciones no periódicas de la tarea sucesora. Por este motivo pueden existir en un intervalo t las $\lceil t/T \rceil$ generaciones de los pares de tareas predecesora-sucesora, más la primera generación de la sucesora que, por generarse aperiódicamente, queda dentro del intervalo t pero no así su predecesora.

5.4 Lema 5.3

Si τ_i es tarea raíz y τ_k es una tarea perteneciente a otro trabajo cuya predecesora tiene menor prioridad que τ_i, entonces podrá existir como máximo una ejecución completa de τ_k, y sus sucesivas sucesoras con mayor prioridad que τ_i, en el intervalo definido por la generación de τ_i y su ejecución completa.

Demostración

La generación de la tarea τ_i puede realizarse únicamente luego de la ejecución completa de la predecesora de τ_k. Sin embargo, no puede haber una nueva ejecución completa de la predecesora de τ_k, por existir un requerimiento de mayor prioridad dado por τ_i. Por este motivo tampoco habrá una nueva generación y correspondiente ejecución.

Según los lemas anteriores, en un sistema que posee tareas con relaciones de precedencia, el peor caso de carga será diferente para tareas raíces de trabajos o tareas con predecesoras.

5.5 Lema 5.4

El máximo tiempo de ejecución de una tarea raíz τ_i, MTR_i, será:

$$MTR_i = \min t | t \geq C_i + \sum_{\forall \tau_h \in \mathbf{R}_i} C_h \left\lceil \frac{t}{T_h} \right\rceil + \sum_{\forall \tau_h \in \mathbf{T}_i} C_h \left(\left\lceil \frac{t}{T_h} \right\rceil + 1 \right) + \sum_{\forall \tau_h \in \mathbf{U}_i} C_h \qquad (5.5)$$

donde

$\mathbf{R}_i = \left\{ \tau_h : \nexists \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_i) \right\}$

$\mathbf{T}_i = \mathbf{A}_i^r | \mathbf{A}_i^r = \mathbf{A}_i^{r+1}$ y $\nexists\, q < r$ tal que $\mathbf{A}_i^r = \mathbf{A}_i^q$, en donde $\mathbf{A}_i^k$ es definido en forma recursiva como

$$\mathbf{A}_i^0 = \left\{ \tau_h : \exists \tau_p \in \mathbf{R}_i | \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_i) \right\}$$

$$\mathbf{A}_i^k = \left\{ \tau_h : \exists \tau_p \in \mathbf{A}_i^{k-1} | \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_i) \right\} \cup \bigcup_{l=0}^{k-1} \mathbf{A}_i^l$$

$\mathbf{U}_i = \mathbf{B}_i^r | \mathbf{B}_i^r = \mathbf{B}_i^{r+1}$ y $\nexists\, q < r$ tal que $\mathbf{B}_i^r = \mathbf{B}_i^q$, en donde $\mathbf{B}_i^k$ es definido en forma recursiva como

$$\mathbf{B}_i^0 = \left\{ \tau_h : \exists \tau_p \prec \tau_h | \left(P(\tau_p) < P(\tau_i) \right) \wedge \left(P(\tau_h) > P(\tau_i) \right) \right\}$$

$$\mathbf{B}_i^k = \left\{ \tau_h : \exists \tau_p \in \mathbf{B}_i^{k-1} | \left(\tau_p \prec \tau_h \right) \wedge \left(P(\tau_h) > P(\tau_i) \right) \right\} \cup \bigcup_{l=0}^{k-1} \mathbf{B}_i^l$$

Demostración:

La ecuación 5.5 determina el tiempo en que se ejecutan todas las generaciones de las tareas, pertenecientes a los conjuntos que intervienen en la sumatoria. Para determinar el máximo tiempo de ejecución de τ_i, hay que considerar el máximo número de generaciones, de tareas de mayor prioridad, que pueden existir entre su generación y su ejecución completa. El conjunto $\mathbf{R}_i$ contiene todas las tareas raíces con mayor prioridad que τ_i. Estas tareas pueden tener como máximo $\left\lceil t/T \right\rceil$ generaciones en un intervalo t según el lema 5.1. El conjunto $\mathbf{T}_i$ contiene todas las tareas que poseen predecesoras con mayor prioridad que τ_i que podrán generarse como máximo $\left\lceil t/T \right\rceil+1$ como se demuestra en el lema 5.2. El conjunto $\mathbf{U}_i$ contiene todas las tareas, con mayor prioridad que τ_i, cuyas predecesoras poseen menor prioridad que τ_i o pertenecen al conjunto $\mathbf{U}_i$. Estas tareas, como lo demuestra el lema 5.3, podrán ejecutarse una sola vez en el intervalo comprendido entre la generación y la ejecución completa de τ_i. El mínimo t en que se ejecuten el máximo número de generaciones posibles más la propia ejecución de τ_i, será el máximo tiempo de ejecución de dicha tarea.

5.6 Lema 5.5

No pueden existir simultáneamente requerimientos pendientes de dos sucesoras τ_j y τ_l, pertenecientes a diferentes trabajos, cuyas correspondientes predecesoras, τ_i y τ_k, tienen menor prioridad que la menor prioridad de ellas. Es decir, τ_j y τ_l no podrán tener requerimientos pendientes simultáneamente si $\text{máx}(P(\tau_i),P(\tau_k)) < \text{mín}(P(\tau_j),P(\tau_l))$

Demostración:

Para que una tarea sucesora se genere, debe ejecutarse completamente su correspondiente predecesora. Si $\text{máx}\ (P(\tau_i),P(\tau_k)) < \text{mín}\ (P(\tau_j),P(\tau_l))$, la

generación de una sucesora imposibilita la ejecución de la predecesora de la otra sucesora, por tener mayor prioridad que ella. De esta manera al no poder ejecutarse la predecesora, no podrá generarse su correspondiente sucesora.

5.7 Lema 5.6

El máximo tiempo de ejecución de una tarea τ_j, MTR_j, sucesora de τ_i con $P(\tau_i)<P(\tau_j)$, será:

$$MTR_j = \min t | t \geq C_j + \sum_{\forall \tau_h \in \mathbf{R}_j} C_h \left\lceil \frac{t}{T_h} \right\rceil + \sum_{\forall \tau_h \in \mathbf{T}_j} C_h \left(\left\lceil \frac{t}{T_h} \right\rceil + 1 \right) \qquad (5.6)$$

donde

$\mathbf{R}_j = \left\{ \tau_h : \exists \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_j) \right\}$

$\mathbf{T}_j = \mathbf{A}_j^r | \mathbf{A}_j^r = \mathbf{A}_j^{r+1}$ y $\nexists\, q < r$ tal que $\mathbf{A}_j^r = \mathbf{A}_j^q$, en donde $\mathbf{A}_j^k$ es definido en forma recursiva como

$$\mathbf{A}_j^0 = \left\{ \tau_h : \exists \tau_p \in \mathbf{R}_j | \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_j) \right\}$$

$$\mathbf{A}_j^k = \left\{ \tau_h : \exists \tau_p \in \mathbf{A}_j^{k-1} | \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_j) \right\} \cup \bigcup_{l=0}^{k-1} \mathbf{A}_j^l$$

Demostración:

La ecuación 5.6 determina el tiempo en que se ejecutan todas las generaciones de las tareas pertenecientes a los conjuntos que intervienen en la sumatoria. El conjunto $\mathbf{R}_j$ contiene todas las tareas raíces con mayor prioridad que τ_j. Estas tareas puede tener como máximo $\left\lceil t/T \right\rceil$ generaciones en un intervalo t según el lema 5.1. El conjunto $\mathbf{T}_j$ contiene todas las tareas que poseen predecesoras con mayor prioridad que τ_i que podrán generarse como máximo $\left\lceil t/T \right\rceil$+1 como se demuestra en el lema 5.2. El mínimo t en que se ejecuten el máximo número de generaciones posibles más la propia ejecución de τ_j será el máximo tiempo de ejecución de dicha tarea.

5.8 Teorema 5.1

Un trabajo será diagramable, si la sumatoria de los *MTR* de las tareas que integran un trabajo, predecesora-sucesora de un tarea raíz a una tarea hoja, es menor que el vencimiento de dicho trabajo.

Demostración

Es inmediata al establecer que el máximo tiempo que demandará la ejecución de un trabajo es el tiempo que lleva las subtareas que lo integran.

5.9 Corolario

Un sistema será diagramable si todos los trabajos que lo integran son diagramables.

Demostración

Inmediata de la definición de sistema de tiempo real.

6. Ejemplo

Supongamos tener un sistema de tres tareas con los siguientes requerimientos de tiempo real:

	C_i	T_i
τ_1	3	6
τ_2	3	8
τ_3	1	8

Si se aplica el método de las ranuras vacías, se determina que el sistema no es diagramable por la disciplina de Períodos Monotónicos Crecientes. Debido a que el sistema cumple con las hipótesis de Liu y Layland ([19]), y dado que la disciplina de Períodos Monotónicos Crecientes es óptima, se puede afirmar que el sistema no es diagramable bajo ninguna pila de Prioridades Fijas. Sin embargo, el sistema tendrá los mismos requerimientos de tiempo real si se proponen las siguientes modificaciones:

	C_i	T_i
τ_{1a}	2	6
τ_{1b}	1	6
τ_{2a}	1	8
τ_{2b}	2	8
τ_3	1	8

Se observa que el tiempo de procesamiento de las trabajos (τ_{1a}, τ_{1b}) y (τ_{2a}, τ_{2b}) son iguales al de las tareas τ_1 y τ_2 respectivamente, con lo que se mantienen las características de tiempo real del sistema original.

Consideremos la siguiente pila de prioridades: {τ_{1b}, τ_{2b}, τ_3, τ_{1a}, τ_{2a}} siendo τ_{1b} la tarea de mayor prioridad.

6.1 Cálculo de MTR_{1a}:

Puesto que τ_{1a} es tarea raíz hay que aplicar el lema 5.4 para determinar su *MTR*. Para τ_{1a} será:

$\mathbf{R}_{1a}=\{\tau_3\}$

$\mathbf{T}_{1a}=\phi$

$\mathbf{U}_{1a}=\{\tau_{2b}\}$

y la ecuación 5.5 tendrá la siguiente forma:

$$MTR_{1a} = \min t | t \geq 2 + 1 \cdot \left\lceil \frac{t}{8} \right\rceil + 2$$

por lo que $MTR_{1a} = 5$.

6.2 Cálculo de MTR_{1b}:

Debido a que τ_{1b} es una tarea sucesora hay que aplicar el lema 5.6 para determinar su *MTR*. Para τ_{1b} será:

$\mathbf{R}_{1b}=\phi$

$\mathbf{T}_{1b}=\phi$

y la ecuación 5.6 tendrá la siguiente forma:

$$MTR_{1a} = \min t | t \geq 1$$

por lo que $MTR_{1b} = 1$.

Dado que el trabajo τ_1 es una cadena de sólo dos tareas, el único camino entre una tarea raíz y una tarea hoja será τ_{1a} - τ_{1b}. Como MTR_{1a} + MTR_{1b} = 6, y aplicando el teorema 5.1, el trabajo será diagramable por ejecutarse antes de su vencimiento (D_1 = 6).

6.3 Cálculo de MTR_{2a}:

Como τ_{2a} es tarea raíz hay que aplicar el lema 5.4 para determinar su *MTR*. Para τ_{2a} será:

$\mathbf{R}_{2a}$={ τ_3, τ_{1a} }

$\mathbf{T}_{2a}$= { τ_{1b} }

$\mathbf{U}_{2a}$= ϕ

y la ecuación 5.5 tendrá la siguiente forma:

$$MTR_{2a} = \min t | t \geq 1 + 1 \cdot \left\lceil \frac{t}{8} \right\rceil + 2 \cdot \left\lceil \frac{t}{6} \right\rceil + 1 \cdot \left(\left\lceil \frac{t}{6} \right\rceil + 1 \right)$$

por lo que MTR_{2a} = 6.

6.4 Cálculo de MTR_{2b}:

Debido a que τ_{2b} es tarea con predecesora hay que aplicar el lema 5.6 para determinar su *MTR*. Para τ_{2b} será:

$\mathbf{R}_{1b}$= ϕ

$\mathbf{T}_{1b}$= ϕ

con lo que la ecuación 5.6 queda:

$$MTR_{1a} = \min t | t \geq 2$$

por lo que MTR_{1b} = 2.

Dado que el trabajo τ_2 es también una cadena de sólo dos tareas, el

único camino entre una tarea raíz y una tarea hoja será τ_{2a} - τ_{2b}. Como $MTR_{2a} + MTR_{2b} = 8$, y aplicando el teorema 5.1, el trabajo será diagramable por ejecutarse antes de su vencimiento ($D_1 = 8$).

6.5 Cálculo de MTR_3:

Como τ_3 es tarea raíz hay que aplicar el lema 5.4 para determinar su *MTR*. Para τ_3 será:

$\mathbf{R}_3 = \phi$

$\mathbf{T}_3 = \phi$

$\mathbf{U}_3 = \{ \tau_{1b}, \tau_{2b} \}$

con lo que la ecuación 5.5 tiene la siguiente forma:

$$MTR_3 = \min t | t \geq 1+1+2$$

por lo que $MTR_3 = 4$.

Dado que el trabajo τ_3 está compuesto por una única tarea, es suficiente que su *MTR* sea menor que su vencimiento para ser diagramable. Como la inecuación $MTR_3 \leq D_3$ ($4 \leq 8$) es satisfecha, entonces τ_3 es diagramable.

Por el corolario del teorema 5.1, el sistema será diagramable por ser diagramables todos los trabajos que lo componen.

7. Características de las relaciones de precedencias en sistemas monoprocesador con prioridades fijas

En este capítulo se ha demostrado una condición suficiente para asegurar la diagramabilidad de un sistema de tiempo real con tareas con relaciones de precedencia. Se comprobó que las relaciones de precedencias poseen un mecanismo implícito de diagramación que, aprovechado convenientemente, permite mejorar la diagramabilidad de la disciplina de

Prioridades Fijas. Para poder aprovechar esta característica de las relaciones de precedencias, es necesario que la prioridad de la tarea sucesora sea mayor que la de sus correspondientes predecesoras.

El peor caso de carga de Liu y Layland para sistemas monoprocesadores y tareas independientes no es válido cuando existen relaciones de precedencia. Esta característica no permite aplicar muchos de los resultados obtenidos en la teoría de tiempo real, por basarse un gran número de ellos en comprobar la diagramabilidad bajo el peor caso de carga.

La no optimalidad de la disciplina de Períodos Monotónicos Crecientes entre las disciplinas de Prioridades Fijas en sistemas con tareas con relaciones de precedencia, hace más complejo determinar la pila de prioridades que hace diagramable al sistema. En sistemas con tareas independientes, sólo es necesario determinar la diagramabilidad para una pila de prioridades ordenada por Períodos Monotónicos Crecientes para asegurar la diagramabilidad del sistema. Debido a que no se demostró formalmente la existencia de un ordenamiento óptimo cuando existen relaciones de precedencia, el análisis de la diagramabilidad de sistemas de tiempo real con estas características es más complejo.

Capítulo 6

En este capítulo se demuestra una condición suficiente para garantizar la diagramabilidad de un sistema multiprocesador de tiempo real duro con tareas con relaciones de precedencia. Se propone un mecanismo de contadores para garantizar la generación periódica de todas las tareas de los trabajos. Este mecanismo permite analizar la diagramabilidad del sistema, utilizando condiciones de diagramabilidad desarrolladas para sistemas monoprocesador.

1. Sistemas Multiprocesador de Tiempo real Duro con Relaciones de Precedencia

Uno de los problemas más complejos en la Teoría de Sistemas Distribuidos es la asignación de un conjunto de tareas de tiempo real en un conjunto de procesadores de forma tal que el sistema resulte diagramable.

El problema de asignación puede ser definido de la siguiente manera: Existe un conjunto de *m* tareas apropiativas de tiempo real y un conjunto de *n* procesadores interconectados por una red de comunicaciones, cada uno de los cuales tiene un determinado número de recursos (poder de procesamiento, memoria, dispositivos de entrada salida, etc.). Las tareas

son periódicas y cada una de ellas tiene un determinado período, vencimiento, tiempo máximo de ejecución y dispositivos necesarios para poder ejecutarse. El sistema se considerará de tiempo real duro. Por lo tanto, ningún vencimiento puede ser perdido si se desea un funcionamiento correcto. La diagramación del sistema se realiza en forma local, por lo que cada tarea debe ejecutarse siempre en el procesador al que fue *asignada*. La determinación del procesador en que se ejecutará la tarea se realiza en la etapa de diseño y permanece invariante en tiempo de corrida. Un conjunto de asignaciones que distribuyen todas las tareas del sistema en todos o parte de los procesadores se denomina *asignación tentativa*. Si no existe ningún tipo de restricción para asignar una tarea en alguno de los procesadores, entonces el número de asignaciones tentativas posibles es n^m. Esto determina que la complejidad es NP-completo al problema de encontrar una asignación tentativa óptima [25]. Se considerará que las tareas tienen cinco diferentes restricciones: *preasignación*, *recursos*, *tiempo real*, *comunicación* y *precedencia*.

2. Restricciones

2.1 Restricciones de preasignación

Cuando una tarea requiere para ejecutarse características o recursos especiales de alguno de los procesadores del sistema (p.e. poder de procesamiento del procesador), entonces la asignación de dicha tarea tiene restricciones de preasignación. Similarmente, cuando se desea tolerancia a fallas, el procesamiento de algunas tareas deben ser duplicadas. Obviamente, la réplica no puede ser asignada al mismo procesador al que fue asignada la tarea original. Esta característica de las tareas replicadas introducen también restricciones de preasignación.

2.2 Restricciones de recursos

Para ser ejecutada, cada tarea requiere la utilización de un determinado número de recursos del procesador (memoria, dispositivos de Entrada/Salida, etc.). Luego de asignar varias tareas a un procesador, alguno de estos recursos pueden ser insuficientes para todas ellas. Por ejemplo, la memoria que requieren todas las tareas asignadas a un procesador, debe ser menor o a lo sumo igual a la memoria que dispone dicho procesador.

2.3 Restricciones de tiempo real

Todas las tareas deben ser ejecutadas antes de sus respectivos vencimientos. Esto determina restricciones de tiempo real a la asignación tentativa. Es necesario verificar la diagramabilidad de todas las tareas asignadas a un procesador, como también la diagramabilidad de todo el conjunto de procesadores. Se considerará una diagramación local, esto es, cada procesador tiene un algoritmo de diagramación y lo aplica a las tareas que le fueron asignadas en el momento de diseño.

2.4 Restricciones de precedencia

Cuando una tarea (sucesora) necesita datos producidos previamente por otra/s tarea/s (predecesora/s), se establece una relación de precedencia. El hecho de que una tarea sucesora no puede ser ejecutada antes de su correspondiente/s predecesora/s, constituye las restricciones de precedencia. En un sistema multiprocesador de tiempo real duro, las relaciones de precedencias introducen un comportamiento particular que debe ser especialmente considerado para garantizar la diagramabilidad del sistema.

2.5 Restricciones de comunicación

En el caso en que la tarea predecesora y la sucesora sean asignados a

diferentes procesadores, los datos producidos por la predecesora deben ser transmitidos a la sucesora a través de la red de comunicaciones. Debe garantizarse que el tiempo necesario para transmitir dichos datos no haga entrar en crisis al trabajo. Esto introduce restricciones de comunicación. Se puede considerar a la red de comunicaciones como un subsistema de tiempo real.

En la mayoría de los casos, exceptuando precedencia y tiempo real, las restricciones anteriores pueden ser verificadas en etapas intermedias del método de asignación. Sin embargo, la comprobación de las restricciones de precedencia y tiempo real, al necesitar conocer la asignación de todas las tareas, sólo es posible realizarla sobre la asignación tentativa completa. En este capítulo se demuestra formalmente un método para validar las restricciones de precedencia y tiempo real de una asignación tentativa, y se propone un mecanismo para evitar la desincronización entre las tareas de un mismo trabajo provocada por las relaciones de precedencia.

3. Asignaciones tentativas. Un método No-Convencional

Diferentes métodos han sido propuestos para hallar asignaciones tentativas para sistemas de tiempo real con distintas restricciones [26, 30, 41]. Un método no convencional, aporte original de esta tesis, consiste en el uso de Lógica Difusa para diseñar un Sistema de Producción por Aprendizaje [24]. En este método de asignación se definen variables difusas cuyo objetivo es evaluar la conveniencia de asignar una tarea a un determinado procesador. Para cada variable son definidas tres funciones de pertenencia: baja, media y alta. Un diseñador experto podría determinar un conjunto de reglas que se basarán en dichas variables, para establecer cuál es la asignación más conveniente. De esta manera podría asegurar que si el ancho de banda disponible, luego de realizar la asignación, es "alto" y la

utilización de los procesadores "no es muy" desequilibrada, entonces es "bastante conveniente" realizar la asignación de la tarea al procesador. Obviamente, las condiciones "alto", "no es muy" y "bastante conveniente" del ejemplo, son fácilmente implementadas por las funciones de grado de pertenencia, mediante variables difusas. El inconveniente es determinar todos los predicados que son necesarios considerar, pues generalmente depende de las particularidades del sistema de tiempo real a resolver (p.e.: restricciones de comunicación muy rigurosas). Para poder estipular las propociciones adecuadas al realizar una asignación tentativa "posiblemente diagramable", en el instante inicial se crean todos los posibles predicados que se generan al combinar las diferentes funciones de pertenencia de las variables difusas. Para la obtención de las reglas difusas se construye una matriz para cada una de las variables difusas, las cuales tienen asociada una fila a cada tarea a asignar (m) y una columna a cada procesador disponible (n). De esta manera se crean tantas matrices de mxn como variables difusas. El elemento (x,y) de la matriz correspondiente a la variable difusa i, es el valor de la variable luego de asignar la tarea asociada a la fila x, al procesador asociado a la columna y. El valor mínimo y máximo de cada variable define el rango de la variable difusa. Luego, si se divide dicho intervalo en tres subintervalos iguales, pueden establecerse sendas funciones trapezoidales de pertenencia a los conjuntos difusos "bajo", "medio" y "alto". De esta manera cada elemento de la matriz tendrá un grado de pertenencia a los conjuntos bajo, medio y alto.

Un Sistema de Producción Difuso está formado por pxq subconjuntos de Reglas de Producción [18]. En este sistema, cada subconjunto está compuesto por 3^k predicados de primer orden, donde k es el número de variables difusas. El consecuente de cada predicado es de la forma "*entonces es bueno asignar la tarea* τ_x *al procesador* P_y". El antecedente de cada predicado estará constituido por la intersección difusa del grado de pertenencia de cada una de las variables. Nótese que el conjunto de 3^k

predicados cubren todas las variaciones con repetición de todas las variables en los conjuntos "bajo", "medio" y "alto". Obviamente habrá *m*x*n* subconjuntos que contendrán 3^k predicados cada uno. Una vez evaluado todos los predicados de los subconjuntos se obtiene el máximo valor difuso de los predicados en cada subconjunto. A este valor se lo denominará *coeficiente de oportunidad*. Al provenir de cada uno de los subconjuntos, cada coeficiente estará asociado a una asignación en particular. Los coeficientes de oportunidad son ordenados en una pila en forma decreciente. Luego, el método realiza la asignación que tiene mayor coeficiente de oportunidad. Si la asignación es no factible, se selecciona la asignación que le sigue en la pila de coeficientes. Una vez que la asignación de una tarea resultó factible, se comienza una nueva iteración del método evaluando nuevamente las matrices de las variables difusas. Luego de cada asignación, las matrices de las variables difusas tendrán una fila menos y habrá *p* subconjuntos de posibles asignaciones menos que la iteración anterior. Si ninguna asignación resulta factible, se deshace la asignación anterior y se reanuda el proceso a partir del siguiente coeficiente de oportunidad en la iteración anterior. El proceso es interrumpido si se alcanza un número predeterminado de intentos de asignación no factibles. Posteriormente, el Proceso de Aprendizaje es el encargado de ponderar cada uno de los predicados. De esta manera, a los predicados que influyeron para realizar una asignación que no resultó diagramable le serán reducido su peso, mientras que a los que produjeron buenas asignaciones le serán aumentados. La resolución de cada predicado se realizará entonces, evaluando el antecedente y multiplicándolo por el peso asociado a dicho predicado.

A los efectos de probar experimentalmente el método, se desarrolló un conjunto de problemas generados aleatoriamente según las pautas descriptas en [26]. Cada problema consiste en tres trabajos de 4, 8 y 12 tareas a ser asignadas a seis procesadores. Las relaciones de precedencias

fueron realizadas mediante un generador aleatorio de grafos. Seis de las tareas fueron preasignadas aleatoriamente. Los períodos del segundo y tercer trabajo fueron el doble y el triple, respectivamente, del período del primero. Los factores de utilización de los trabajos fueron especificados en 0,5 por ser el máximo que el autor utiliza en [26]. La longitud de los mensajes entre las tareas fueron aleatoriamente asignados entre 50 y 450 bytes. El ancho de banda de la red de comunicaciones fue limitado a 100 bytes/mseg. El tiempo promedio necesario para encontrar una solución fue de 4,9 segundos, el promedio del número de intentos de asignación no factibles fue 18,6 y el promedio del ancho de banda de la red de comunicaciones para cada asignación que resultó factible fue 31 bytes/mseg.

El método descripto puede ser utilizado para obtener asignaciones tentativas que cumplen con todas las constricciones (tiempo, preasignación, recursos, etc.) excepto las de precedencia. Si se verifica que también éstas se cumplen, la asignación tentativa ha sido validada y se obtuvo una solución al problema.

El Sistema de Producción por Aprendizaje es, por lo tanto, una herramienta que prepara el sistema para ser validado mediante los procedimientos que se demuestran formalmente en lo que sigue. Esa preparación puede lograrse también con otras herramientas (heurísticas, recocido simulado, etc.). El Sistema de Producción por Aprendizaje fue presentado como herramienta auxiliar por ser un resultado original del autor.

4. Restricciones de Precedencia: Validando una Asignación Tentativa.

En [7, 44] han sido analizados sistemas multitarea-monoprocesador con tareas con relaciones de precedencia. Cuando las tareas que se comunican están en diferentes procesadores, la tarea sucesora puede generarse cuando

recibe el último bit de los datos transmitidos por su predecesora. El tiempo necesario para transmitir los datos de una predecesora a una sucesora a través de la red de comunicaciones será simbolizado Δ. Las tareas que no poseen sucesora son denominadas *hojas*.

Aunque no explícitamente diferenciadas, las trabajos de investigación de tiempo real para sistemas multiprocesadores, consideran dos tipos de precedencias. El primer tipo de precedencia es utilizado, por ejemplo, en la solución al problema presentado en [41]. En dicho ejemplo, es tácitamente asumido que la generación de la tarea sucesora puede diferir, de la generación de su predecesora, en un período del trabajo al que pertenecen. Este tipo de precedencia será denominada *precedencia blanda*. La figura 6.1 muestra la diagramación de un par de tareas predecesora-sucesora con precedencias blandas. La primera generación de la tarea sucesora se produce en el segundo período del trabajo luego de recibir los datos producidos por la primera generación de la tarea predecesora. La validación de una asignación tentativa se reduce a verificar que la ejecución de la tarea y, si corresponde, la completa transmisión de los datos a su sucesora, se realiza antes de sus correspondientes vencimientos. Para esto se analiza la diagramabilidad de cada procesador, pero se modifican los vencimientos de cada una de las tareas asignadas a él, restándoles, cuando correspondiere, el tiempo de transmisión de los datos a la sucesora. En el caso de precedencias blandas el vencimiento del trabajo tendrá que ser menor o a lo sumo igual al máximo número de pares de tareas predecesora-sucesora que existe entre una tarea raíz y una tarea hoja.

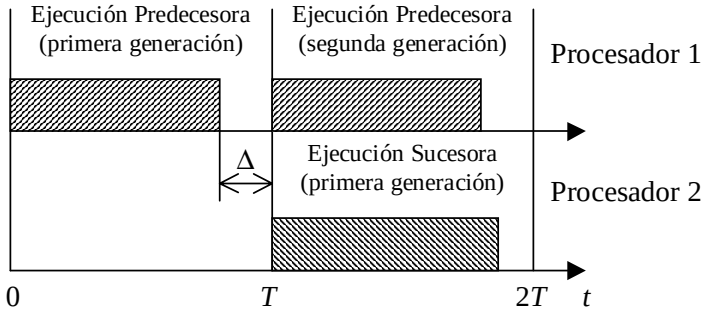


Figura 6.1

Cuando el vencimiento de un trabajo es menor o igual que su período deberá utilizarse una diagramación con *precedencias duras*. Si un sistema es diagramado con precedencias duras, la ejecución de todas las tareas debe realizarse completamente antes de la próxima generación del trabajo. Este tipo de precedencias es el más común en sistemas de tiempo real y es el que se considera en esta tesis.

5. El Modelo del sistema

El sistema está compuesto por p procesadores que deben ejecutar n trabajos. El trabajo j está integrado por $m(j)$ tareas, numeradas τ^j_1, τ^j_2,..., $\tau^j_{m(j)}$. El conjunto de n trabajos está completamente especificado por $\mathbf{J} = \{(T_1, D_1, \Lambda_1), (T_2, D_2, \Lambda_2), ..., (T_n, D_n, \Lambda_n)\}$ donde T_j, D_j y Λ_j representan el tiempo mínimo entre generaciones, el vencimiento y el conjunto de tareas que componen el trabajo j respectivamente. El conjunto de $m(j)$ tareas que integran el trabajo j está completamente especificado por $\Lambda_j = \{(C^j_1, \Phi^j_1), (C^j_2, \Phi^j_2),..., (C^j_{m(j)}, \Phi^j_{m(j)})\}$ donde C^j_i y Φ^j_i representan el máximo tiempo de ejecución de la τ_i y el conjunto de sus tareas predecesoras respectivamente.

Cada tarea está asignada a un solo procesador. Tareas del mismo trabajo pueden estar asignadas a distintos procesadores. Cada procesador tiene un conjunto de tareas asignadas. La diagramación es local, por lo que en cada procesador existe un diagramador y sólo actúa sobre las tareas asignadas a dicho procesador.

Una tarea puede generarse sólo una vez que hayan terminado de ejecutarse todas sus predecesoras. La terminación de todas las tareas predecesoras produce el arribo de la sucesora. La generación de las tareas que no poseen predecesoras ($\Phi = \varnothing$), denominadas *raíces*, coincide con la del trabajo al que pertenecen.

El camino λ^j_i es definido por una secuencia de tareas τ^j_1, τ^j_2,..., τ^j_i,

pertenecientes al trabajo *j*, tal que: τ^j_1 es tarea raíz y $\tau^j_h \prec \tau^j_{h+1}$ para $1 \leq h < i$. Λ^j_i simbolizará el conjunto de todos los caminos del trabajo *j*, que tienen como origen una tarea raíz y finalizan en la tarea τ^j_i.

5.1 Definición 6.1

Será denominada *Máximo Tiempo de Respuesta*, simbolizada MTR^j_i, a una cota del máximo número de ranuras que puede existir entre la generación de la tarea τ^j_i y el final de su completa ejecución.

Esto significa que, si el diagramador no deja ninguna ranura libre cuando existen tareas que requieren el procesador, la ejecución completa de la tarea τ^j_i se realizará siempre dentro del intervalo de *MTR* ranuras que comienza en la generación de dicha tarea.

5.2 Definición 6.2

Se denomina *Máximo Tiempo de Generación* de τ^j_i, notado MTG^j_i, a la máxima cantidad de ranuras que la tarea podrá tener desplazada su generación con respecto a la generación del trabajo.

Obviamente, el *MTG* de toda tarea raíz es cero.

Si la tarea τ^j_i se genera MTG^j_i ranuras luego de la generación del trabajo *j*, entonces, al ser T^j el número mínimo de ranuras entre generaciones de dicho trabajo, la tarea τ^j_i tendrá al menos T^j ranuras entre dos generaciones consecutivas. El mecanismo de activación temporizada, descripto más adelante, es el encargado de que la tarea τ^j_i se genere MTG^j_i ranuras después de la generación del trabajo *j*. Con estas consideraciones, las tareas asignadas a cada uno de los procesadores cumplen con las condiciones de Liu y Layland [19]. De esta manera el método de las ranuras vacías [31], desarrollado para sistemas multitarea-monoprocesador, puede aplicarse para determinar la diagramabilidad de las tareas asignadas a cada

uno de los procesadores del sistema.

Como cada tarea produce su generación luego de *MTG* ranuras de la generación de su correspondiente trabajo, existirá una sincronización entre todas las tareas que integran dicho trabajo. Esta sincronización permitirá establecer intervalos relativos entre las generaciones de tareas pertenecientes al mismo trabajo.

5.3 Definición 6.3

$\mathbf{L}^j_i$ es definido como la unión de dos conjuntos de tareas asignadas al procesador en donde τ^j_i es asignada. El primero de ellos es el conjunto de tareas que, no perteneciendo al trabajo *j*, tienen una prioridad mayor que τ^j_i. El segundo conjunto contiene las tareas que perteneciendo al trabajo *j*, tienen mayor prioridad que τ^j_i y deben ejecutarse en el intervalo en donde τ^j_i debe ejecutarse. Esto es, el intervalo de duración MTR^j_i que comienza MTG^j_i ranuras luego de la generación del trabajo.

5.4 Lema 6.1

El MTR^j_i estará dado por:

$$MTR_i^j = \min t \mid t = C_i^j + \sum_{\forall h,k \mid \tau_h^k \in \mathbf{L}_i^j} C_h^k \cdot \left\lceil \frac{t}{T_h^k} \right\rceil \qquad (6.1)$$

Demostración

Las tareas que estarán en condiciones de ser ejecutadas en procesador, utilizando un algoritmo de Prioridades Fijas, son:

- las que no pertenecen a otros trabajos y tienen una prioridad mayor que τ^j_i. Estas tareas se considerarán, para el cálculo de MTR^j_i, independientes. Esta consideración es pesimista puesto que las tareas que poseen relaciones de precedencia están sincronizadas entre sí, imposibilitando una generación simultánea de las mismas. Esta consideración, a costa de obtener una

condición de diagramabilidad pesimista, permite utilizar los métodos de validación desarrollados para sistemas monoprocesador.

- las que perteneciendo al trabajo *j*, tienen mayor prioridad que τ^j_i y su *MTG* se encuentra en el intervalo en donde τ^j_i debe ejecutarse. Aunque la tarea no se genere simultáneamente con la tarea τ^j_i, la carga que produce es la misma debido a que, por tener el mismo período, puede producir un solo requerimiento en todo intervalo si el sistema es de diagramación factible.

Las tareas que pueden ejecutarse en el intervalo donde la tarea τ^j_i requiere el procesador, conforman el conjunto $\mathbf{L}^j_i$.

Considerando las tareas de otros trabajos independientes entre sí, y que la generación de las tareas se produce periódicamente, se satisfacen las condiciones de Liu y Layland [19], para garantizar que el peor caso de carga es la generación simultánea de todas las tareas con mayor prioridad que τ^j_i. En [31], el método de las ranuras vacías determina la máxima cantidad de ranuras que la tarea τ_i puede estar requiriendo el procesador, desde su generación hasta su completa ejecución. Por [5], y siendo $\mathbf{L}^j_i$ el conjunto de tareas que pueden generarse en el intervalo en que la tarea τ^j_i requiere el procesador, entonces

$$MTR^j_i = \min t \mid t = C^j_i + \sum_{\forall h,k \mid \tau^k_h \in \mathbf{L}^j_i} C^k_h \cdot \left\lceil \frac{t}{T^k_h} \right\rceil$$

Cuando la tarea predecesora y sucesora son ejecutadas en distintos procesadores, los datos de la predecesora deben ser transmitidos a través de una red de comunicaciones. La expresión $\Delta^j_{i,h}$ simbolizará el retardo, inherente a la red de comunicaciones, que existirá en la transmisión de los datos de la tarea τ^j_i a la tarea τ^j_h. Si ambas tareas están asignadas al mismo procesador, dicho retardo se considerará nulo.

5.5 Lema 6.2

El máximo tiempo de generación de la tarea τ^j_i, MTG^j_i, será:

$$MTG_i^j = \max\left\{ \sum_{\forall h,k \mid \tau_k^j \in \lambda_l^j \wedge \tau_k^j \prec \tau_h^j} \left(MTR_k^j + \Delta_{k,h}^j \right) \right\} \ \forall\ \lambda_l^j, \Lambda_l^j, \tau_l^j \mid \ \tau_l^j \in \Phi_i^j$$

Demostración:

La generación de una tarea se produce cuando todas sus predecesoras terminan su ejecución y le transmiten sus correspondientes datos. La máxima sumatoria de los tiempos de ejecución completa (*MTR*) y de retardo de comunicaciones (Δ) de cada tarea de los caminos que conducen, desde una tarea hoja a una tarea predecesora, determinará el máximo número de ranuras que tendrá desplazada su generación con respecto a la generación del trabajo. La máxima sumatoria será el camino más lento de todo el grafo que va desde una tarea hoja a la tarea predecesora que, con la finalización de sus datos, produce la generación de τ^j_i.

5.6 Definición 6.4

El tiempo máximo de ejecución de todos los caminos que concluyen en la tarea τ^j_i inclusive, simbolizado TEC^j_i, es definido:

$$TEC_i^j = \max\left\{ \sum_{\forall h,k \mid \tau_k^j \in \lambda_l^j \wedge \tau_k^j \prec \tau_h^j} \left(MTR_k^j + \Delta_{k,h}^j \right) \right\} \ \forall \lambda_l^j \in \Lambda_i^j \qquad (6.2)$$

5.7 Teorema 6.1

El trabajo *j* será diagramable si para toda tarea hoja τ^j_i se cumple:

$$D_j \geq \max \{\ TEC_i^j\ \}\ \forall \tau_i^j \mid \tau_i^j hoja$$

Demostración

El trabajo se ejecutará antes de su vencimiento si todos los caminos hasta las tareas hojas se ejecutan antes del vencimiento del trabajo.

5.7.1 Corolario

El sistema será factible con el diagramador propuesto si para todos los trabajos del sistema se cumple el Teorema 6.1.

5.8 Ejemplo

Para ilustrar el método de validación de una asignación tentativa, se utiliza el problema multitarea-multiprocesador propuesto por Tindell *et al* en [41]. El sistema está compuesto por 43 tareas que forman 11 trabajos que deben ser asignados a 8 procesadores. Algunas tareas son preasignadas a determinados procesadores, mientras que algunos pares de tareas no pueden ser asignadas al mismo procesador.

El conjunto de tareas es completamente especificado en la tabla **6.I**, en la que se definen los tiempos de ejecución, períodos, preasignaciones, memoria y utilización de la red de comunicaciones requerida (50/1 y 52/2 en la primera fila significa que la tarea τ^0_0 debe enviar 50 y 150 bytes a las tareas τ^0_1 y τ^0_2, respectivamente).

Tarea	*T*	*C*	Memoria	Comunicación	Preasignación
0	60	4	3000	50/1; 150/2	0
1	60	4	1500	60/3; 70/4; 30/5	
2	60	2	1200	20/3	
3	60	2	1700		1
4	60	2	3000	60/6	
5	60	4	3000	80/6	
6	60	6	1100		2
7	35	2	500	40/8	1
8	35	2	700		1
9	35	8	900	90/11	0
10	35	14	2200	250/11	
11	35	4	1000		1
12	14	2	1000	150/13; 150/14	2
13	14	2	1500	50/15	
14	14	2	1600	50/15	
15	14	2	1300		3
16	14	2	1100	50/17	3
17	14	2	1000		2
18	35	1	1000	50/19	1
19	35	1	1600		1
20	14	1	1900	40/21	
21	14	2	2000		3
22	14	1	1000	40/23	

23	14	1	2000	40/24	
24	14	1	1000	20/25	
25	14	1	2000	20/26	
26	14	2	7000	20/27; 20/28	
27	14	1	1100	50/29	
28	14	1	900	30/29	
29	14	1	500		6
30	14	1	600	50/31	7
31	14	2	800	70/32	
32	14	2	1300		7
33	20	3	1000	50/35	2; 3
34	20	2	1000	50/35	0; 1
35	20	2	1000	60/36; 60/37	
36	20	2	1000		6; 7
37	20	2	1000		
38	20	3	1000	50/40	2; 3
39	20	2	1000	50/40	0; 1
40	20	2	1000	60/41; 60/42	
41	20	2	1000		6; 7
42	20	2	1000		

Tabla 6.**I**

En la tabla **6.II** se especifica la capacidad de memoria de cada procesador.

Procesador	Memoria
0	10,000
1	10,000
2	10,000
3	12,000
4	7,000
5	7,000
6	12,000
7	10,000

Tabla 6.**II**

En la figura 6.2 se representan los 11 trabajos compuestos por las 43 tareas del sistema. El número del nodo indica la tarea del trabajo correspondiente, mientras que los pesos asociados a los arcos indican los bytes que se comunican las tareas. Las tareas preasignadas son indicadas por el cuadrado que contiene el procesador correspondiente

Debido a que el problema propuesto por Tindell *et al.* en [41] fue desarrollado considerando precedencias blandas, es muy difícil, tal vez imposible, hallar una asignación diagramable con precedencias duras. Si se

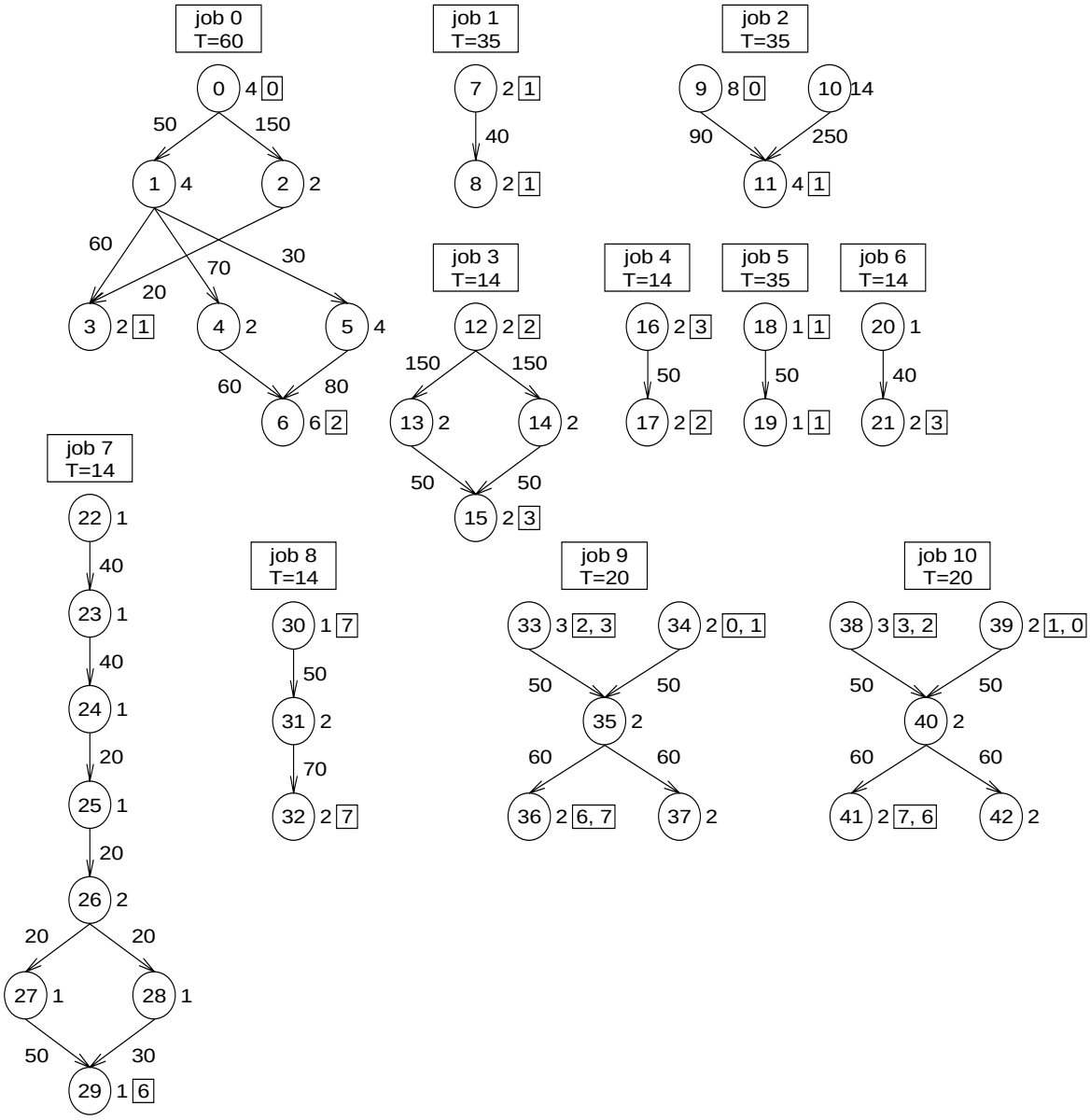


Figura 6.2

observa las capacidades de memoria de los procesadores y las preasignaciones de las tareas, se puede comprobar que tareas con relaciones de precedencia deben ser asignadas a diferentes procesadores. De esta manera, para determinar la diagramabilidad del sistema es necesario considerar el tiempo de transmisión entre las tareas que, por muy alta que

sea la velocidad de la red de comunicaciones, será al menos de una ranura de duración. Estas ranuras. utilizadas para la transmisión de datos, hacen que principalmente los trabajos 4, 8, 9 y 10 sean no diagramables. Una solución a este inconveniente, que permitiría considerar este problema con precedencias duras, sería definir una ranura de tiempo más pequeña. Para esto se deberían calcular nuevamente todos los períodos de generación, vencimientos y tiempos de ejecución de cada una de las tareas del sistema. Dicha solución alteraría completamente el problema planteado por Tindell *et al.*, por lo que no será aplicada.

Para determinar la diagramabilidad del ejemplo propuesto por Tindell *et al.*, se despreciará el tiempo de transmisión en la red de comunicaciones. Esta opción conserva las características de tiempo real del sistema planteado en [41].

En la tabla **6.III** se muestra una asignación tentativa. Las tareas que están remarcadas indican que tienen restricciones de preasignación. Las tareas están ordenadas por períodos monotónicos crecientes.

Procesador	Tareas
0	**34**; **9**; **0**; 1; 2
1	**39**; **18**; **19**; **7**; 10; **8**; **11**; **3**
2	**17**; **12**; **33**; 5; **6**
3	**38**; **15**; 20; **21**; 40; **16**; 42
4	13; 4; 22; 37
5	23; 24; 25; 27; 28
6	26; **29**; 35; **36**
7	14; **30**; 31; **32**; **41**

Tabla 6.**III**

Con toda esta información, aplicando las ecuaciones 6.1 y 6.2, se determinan el *MTR* y el *TEC* de cada tarea. La tabla 6.**IV** contiene los valores calculados para cada una de las tareas. Como puede observarse, ningún trabajo tiene un tiempo de ejecución de camino mayor que su respectivo período. Por lo tanto, el sistema, al cumplir con el corolario del teorema 6.1, es diagramable, con el método propuesto, considerando

precedencia duras.

Tarea	*T*	*MTR*	*TEC*
0	60	14	14
1	60	14	28
2	60	16	30
3	60	30	60
4	60	4	32
5	60	11	39
6	60	13	52
7	35	6	6
8	35	20	26
9	35	10	10
10	35	20	20
11	35	12	32
12	14	4	4
13	14	2	6
14	14	2	6
15	14	5	11
16	14	12	12
17	14	2	14
18	35	3	3
19	35	3	6
20	14	6	6
21	14	7	13

Tarea	*T*	*MTR*	*TEC*
22	14	5	5
23	14	1	6
24	14	1	7
25	14	1	8
26	14	2	10
27	14	1	11
28	14	2	12
29	14	1	13
30	14	3	3
31	14	4	7
32	14	4	11
33	20	7	7
34	20	2	2
35	20	5	12
36	20	5	17
37	20	7	19
38	20	3	3
39	20	2	2
40	20	7	10
41	20	9	19
42	20	9	19

Tabla 6.**IV**

6. Sincronización de Tareas en Sistemas Multiprocesador con Relaciones de Precedencia: Mecanismo de Generación Temporizada

Los métodos para análisis de diagramación propuestos en [11, 15, 19, 31] se basan en la hipótesis de que las tareas son periódicas. Esta hipótesis es válida para los sistemas de tiempo real en que la tareas que lo integran son independientes. El instante en que la tarea recibe los datos de su predecesora será denominado *arribo*. Recién cuando la tarea recibe los

datos de todas sus sucesoras está en condiciones de ejecutarse. Esto no significa que necesariamente deba ser considerada por el diagramador para ejecución. Si la tarea es considerada para ejecución en el arribo de los datos de su última predecesora, no podrá garantizarse un intervalo mínimo entre generaciones consecutivas. Si esto sucede, al no satisfacerse la hipótesis de generación periódica, y al no ser válido el peor caso hallado por Liu & Layland [19], no podrá utilizarse el método de las ranuras vacías para analizar la diagramabilidad de cada uno de los procesadores que integran el sistema.

La recepción de los datos por la sucesora depende del instante en que su predecesora completa su ejecución y del retardo introducido por la red de comunicaciones. El instante en que completa su ejecución la tarea predecesora depende de la secuencia en que requieran la utilización del procesador las tareas asignadas junto que ella. Además, el retardo de la red de comunicaciones que interconecta los procesadores del sistema no es constante (sólo es acotado superiormente), y depende del estado de la red cuando los datos de la predecesora están listos para ser transmitidos. Si, por ejemplo, la red de comunicaciones es del tipo *anillo con ficha* (802.5), los mensajes pueden ser generados cuando la ficha es recibida por el procesador en que se encuentra la tarea predecesora o cuando la ficha es cedida al procesador siguiente. En el primer caso el retardo de transmisión será muy pequeño mientras que en el segundo podrá llegar a ser la cota superior de retardo, simbolizada *UBD* (upper bound delay). Ambos efectos producen una desincronización entre las tareas que componen el trabajo. Cuando esta desincronización es tenida en cuenta, el análisis de la diagramabilidad del sistema se denomina *holístico*[41].

La desincronización producida por las relaciones de precedencias puede ser resuelta si el diagramador cuenta con un mecanismo para detectar y controlar el arribo temprano de datos de las predecesoras [23]. De esta manera, la generación de la tarea sucesora puede ser pospuesta hasta que se

cumpla con el intervalo adecuado relativo a la generación anterior.

El mecanismo propuesto es un contador asociado a cada tarea que se incrementa con una frecuencia de una unidad por ranura. Cuando una tarea raíz es generada (en el instante en que genera el trabajo), el valor de su contador es inicializado en uno y comienza a incrementarse en una unidad por ranura. Cuando la tarea raíz transmite los datos a su(s) sucesora(s), también transmite un valor, denominado *Release Value* y simbolizado *RV*, el cual es 1 más el valor de su contador menos su *MTR* menos, si los datos son transmitidos a través de la red de comunicaciones, el *UBD*. Un *RV* negativo o cero indicará que los datos arribaron antes de que transcurra el máximo tiempo de generación de la tarea sucesora. El contador de la tarea sucesora es cargado con el *RV* de la tarea predecesora. Desde el momento que dicho valor es cargado en el contador, éste es incrementado a razón de una unidad por ranura. La generación de la tarea se produce cuando el contador alcanza el valor 1. Su contenido, sin embargo, continúa incrementándose hasta completar la transmisión del mensaje. La tarea sucesora de una tarea raíz puede, a su vez, ser predecesora de otra tarea a la cual transferirá los datos correspondientes y su *RV*. El procedimiento es repetido para cada par de tareas predecesora-sucesora. El máximo valor posible a ser transferido es 1 y ocurre cuando la tarea predecesora necesita *MTR* ranuras para ejecutarse completamente y el retardo de comunicaciones es de *UBD* ranuras.

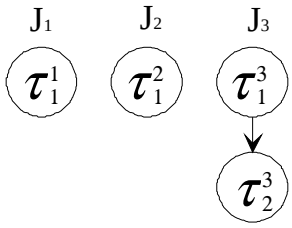
En sistemas donde existen relaciones de precedencia entre las tareas, una tarea sucesora puede poseer varias predecesoras. En este caso, la sucesora debe recibir los datos de todas sus predecesoras para generarse. Cuando los datos y el *RV* de una tarea predecesora arriban, el *RV* es cargado en el contador de la tarea sucesora si su valor es menor que el valor actual del contador. Cuando los datos de todas las predecesoras han sido recibidos y el contador alcanza el valor 1, la tarea sucesora es generada.

Este mecanismo asegura que la generación de cada una de las tareas del trabajo se producen en el máximo tiempo de generación luego de la generación del trabajo. De esta manera, el intervalo entre generaciones se mantiene inferiormente acotado, por lo que las condiciones para aplicar el método de las ranuras vacías son satisfechas.

6.1 Ejemplo: Diagramación por Generación Temporizada.

Para ilustrar conceptualmente el mecanismo se ejemplifica a continuación, un mismo conjunto de trabajos con distintas condiciones de generación y ejecución. El retardo producido por la red de comunicaciones se considerará nulo.

Sean tres trabajos, J_1, J_2 y J_3 y dos procesadores p_1 y p_2. J_1 y J_2 constan de una tarea cada uno (τ^1_1 y τ^2_1 respectivamente). J_3 está formado por dos tareas con relación de precedencia ($\tau^3_1 \prec \tau^3_2$). En las tablas 6.V y 6.VI se consignan los parámetros temporales de los trabajos y de las tareas, como así también sus Λ y Φ.



Trabajo	T	D	Λ
J_1	6	6	$\{\tau^1_1\}$
J_2	8	8	$\{\tau^2_1\}$
J_3	8	8	$\{\tau^3_1, \tau^3_2\}$

Tabla **6.**V

Tarea	C	Φ	MTR
τ^1_1	3	$\varnothing$	3
τ^2_1	6	$\varnothing$	8
τ^3_1	3	$\varnothing$	6
τ^3_2	2	τ^3_1	2

Tabla 6.VI

τ^1_1 y τ^3_1 son asignadas a p_1, en ese orden de prioridad. τ^3_2 y τ^2_1 son asignadas a p_2, en ese orden de prioridad.

La figura 6.3 muestra un diagrama de tiempos de la diagramación del sistema, sin considerar la desincronización de las tareas con relaciones de

precedencia. Se supone que la tarea τ^2_1 genera en t=7, simultáneamente con la tarea τ^3_2. Como puede observarse, en t=15, τ^2_1 pierde su vencimiento y el sistema no cumple con sus restricciones de tiempo real. La no diagramabilidad del sistema proviene de la generación de τ^2_1 en un instante distinto de t=1 con lo que, el tiempo entre generaciones de τ^3_2 es 5 en lugar de 8 que corresponde al período del trabajo J_3. Este simple contraejemplo alcanza para probar que, para sistemas multiprocesador con tareas con relación de precedencia, el peor estado de carga no necesariamente es la generación simultánea de todos los trabajos, como sucede en los sistemas estudiados por Liu y Layland.

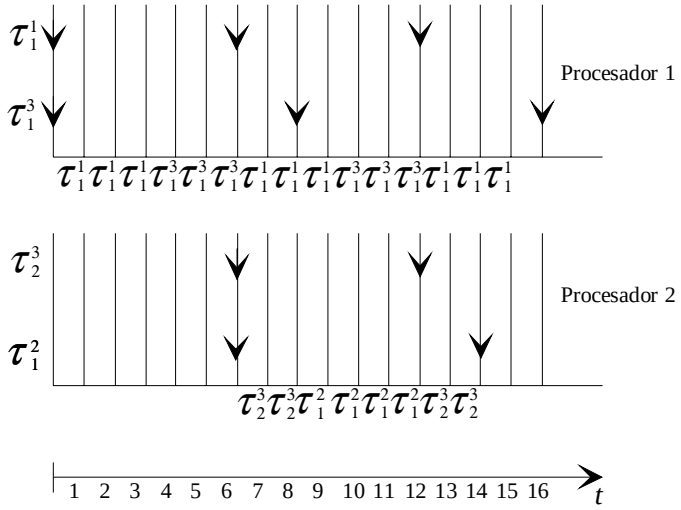


Figura 6.3

Analicemos el sistema aplicando el mecanismo de diagramación temporizada. La figura 6.4 muestra la evolución del sistema, con el valor de los contadores de las tareas en cada ranura. En t=12, el diagramador del procesador p_2, en lugar de conmutar a τ^3_2, sigue ejecutando τ^2_1. En t=14, ésta concluye y a *posteriori* se ejecuta τ^3_2, cumpliendo ambas con sus restricciones de temporales.

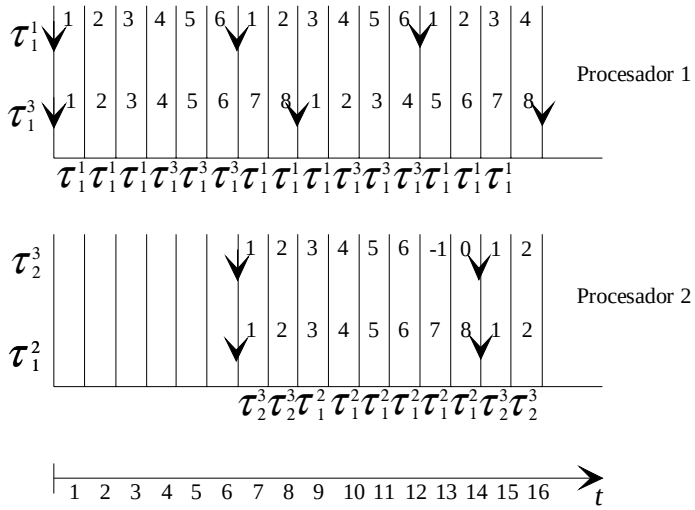


Figura 6.4

Nótese que, en el ejemplo, si a τ^2_1 se le asigna mayor prioridad que a τ^3_2, a pesar de tener períodos idénticos el sistema no resulta diagramable. Esto se debe a la relación de precedencia de τ^3_2 con τ^3_1, que adelanta el vencimiento de la primera para poder cumplir con el vencimiento de J_3.

7. Análisis de la diagramabilidad en Sistemas Multiprocesador: Obtención de una Condición Necesaria.

Existen dos dificultades muy complejas en la teoría de sistemas de tiempo real multitarea-multiprocesador que son: la diagramación y la asignación de tareas. El problema de asignación puede ser descripto de la siguiente manera: existe un conjunto de *m* tareas apropiativas de tiempo real que deben ser ejecutadas en un conjunto de *n* procesadores heterogéneos interconectados a través de una red de comunicaciones. Los procesadores pueden estar diferenciados por su velocidad de procesamiento, capacidad de memoria, capacidad de entradas salidas, recursos especiales, etc. Las tareas están caracterizadas por su tiempo de ejecución (*C*), mínimo tiempo entre generaciones (*T*), vencimiento (*D*) y necesidad de recursos.

La distribución de las *m* tareas en todos o parte de los *n* procesadores

se denomina *asignación*. En [25] se determina que en un sistema multitarea-multiprocesador con m tareas y n procesadores existen n^m diferentes asignaciones posibles. Para visualizar este resultado puede considerarse el siguiente árbol de asignaciones: la tarea 1, representada en la raíz del árbol, puede ser asignada a cualquiera de los n procesadores representados por cada uno de los arcos que salen de la tarea 1. Los m niveles del árbol corresponden a las m tareas a asignar. En cada nivel i se representan las diferentes posibles asignaciones de la tarea τ_i, considerando las asignaciones de las tareas anteriores. De esta manera, cada hoja del árbol representará una posible asignación.

En [25], al considerar tareas independientes, la asignación óptima, con un diagramador con Prioridades Fijas en cada uno de los procesadores, es con la pila de prioridades por Períodos Monotónicos Crecientes. Cuando entre las tareas existen relaciones de precedencia, como se describió anteriormente, la ordenación Períodos Monotónicos Crecientes deja de ser la óptima entre las disciplinas de Prioridades Fijas. De esta manera, el método de asignación debe determinar la prioridad de cada tarea en el procesador que está asignada. Esto aumenta considerablemente la complejidad de hallar una asignación diagramable.

Si el orden de las tareas en los diferentes niveles del árbol determina la prioridad de cada una de las tareas en los procesadores, existirán $m!$ permutaciones posibles de los niveles del árbol. Bajo estas condiciones, el problema de asignación, considerando todas las pilas de prioridades posibles, queda acotado superiormente por $n^m.m!$ asignaciones tentativas. Esto aumenta, si bien continúa siendo NP-completo, la complejidad de obtener una asignación diagramable en un sistema de tiempo real multitarea-multiprocesador.

Existen diversas heurísticas y simulaciones ([24, 26, 30, 41]) que tratan el problema de asignación con diversas restricciones. Una vez obtenida una asignación tentativa, debe verificarse que satisfaga todas las restricciones

del sistema de tiempo real. Esta verificación puede hacerse una vez que se posee dicha asignación tentativa. Sin embargo, pueden establecerse condiciones suficientes para asegurar que ninguna de las n^m posibles asignaciones cumplirá con alguna de las restricciones que impone el sistema. En esta sección se describe una condición suficiente que permite asegurar que ninguna de las posibles asignaciones cumple con las restricciones de diagramación. Se establecen reglas que permiten reducir el conjunto de tareas a asignar y, de esta manera, disminuir la complejidad de determinar que no existe asignación diagramable. Cabe señalar que si se puede obtener un subconjunto reducido de **S**(m) que permita determinar la inexistencia de una asignación diagramable de las m tareas en los n procesadores, la complejidad para determinar la no diagramabilidad del sistema disminuye considerablemente.

7.1 Lema 6.3

El menor MTR^j_i que podrá tener la tarea τ^j_i es su máximo tiempo de ejecución C^j_i.

Demostración:

Por el lema 6.1, el mínimo MTR^j_i estará dado cuando **L** = ϕ. En consecuencia, el t que satisface la ecuación 6.1 es $t = C^j_i$.

7.2 Teorema 6.2

El mínimo tiempo de ejecución que puede tener el trabajo j, simbolizado MTT^j, en su peor caso de ejecución, estará determinado por el máximo TEC^j_i de todas las tareas τ^j_i hojas de dicho trabajo, considerando $MTR^j_i = C^j_i$.

$$MTT^j = \max \sum_{\forall \tau_k \in \lambda^j_i} C^j_k \qquad \forall \tau^j_i \text{ tarea hoja}$$

Demostración:

Por el lema 6.3 el máximo tiempo de respuesta de una tarea τ^j_i es su máximo tiempo de ejecución C^j_i. Considerando esta cota mínima, el camino más lento que finalice en una tarea hoja determinará el menor tiempo en que se que podrá ejecutar completamente el trabajo *j*.

7.3 Corolario

El trabajo *j* será intrínsecamente no diagramable si $D^j > MTT^j$.

La condición anterior, permite disminuir el número de tareas a considerar en el análisis de la diagramabilidad en sistemas multitarea-multiprocesador de tiempo real. Esta reducción del número de tareas puede disminuir considerablemente la complejidad del análisis.

Capítulo 7

En este capítulo se plantea un isomorfismo entre sistemas multitarea-multiprocesador y redes de conmutación de paquetes *multihop*. Este isomorfismo permite utilizar el método de análisis de diagramabilidad propuesto en el capítulo 6 en redes de conmutación de paquetes multihop en tiempo real. El mecanismo de generación temporizada es propuesto para garantizar un intervalo mínimo entre dos generaciones consecutivas.

1. Introducción a redes de conmutación de paquetes

Una red de conmutación de paquetes multihop es una red de comunicaciones orientada a conexión con una topología arbitraria, consistente en un conjunto de *estaciones* interconectadas a través de *enlaces*. Sobre un enlace físico pueden establecerse varias conexiones lógicas. Los mensajes son transportados desde la estación origen a la estación destino utilizando las estaciones intermedias, estableciendo conexiones entre ellas. El conjunto de conexiones que transportan el mensaje desde la fuente al destino es denominado *ruta*.

En [4],[6] y [34] se proponen métodos para establecer rutas tentativas en un sistema, integrado por un conjunto de *m* llamadas de tiempo real con períodos, vencimientos extremo a extremo y longitud de mensajes definidos. Análogamente a lo que ocurre en sistemas multitarea-multiprocesador al establecer una asignación tentativa, en sistemas de conmutación de paquetes multihop es necesario verificar que las restricciones de tiempo real son satisfechas para cada llamada del sistema. El vencimiento extremo a extremo de cada llamada debe ser satisfecho teniendo en cuenta los retardos producidos en los enlaces que conforman la ruta.

2. Definición del sistema

Consideramos una red de topología arbitraria en la que podemos distinguir los siguientes tipos de entidades involucradas: *estaciones*, *clientes* en las estaciones de la red, y la *red* propiamente dicha. Los clientes hacen uso de los servicios de red estableciendo una *llamada* del cliente origen al cliente destino. Cada llamada a nivel cliente es transformada por los servicios de red en una *conexión*. Para cada llamada, la red selecciona una *ruta* entre la estación del cliente origen y la del cliente destino utilizando un conjunto de *enlaces* entre las estaciones de la red, estableciendo conexiones sobre dichos enlaces. Una llamada tiene lugar sobre un conjunto de enlaces mientras que una conexión existe sobre un solo enlace. Luego, por la característica periódica de los mensajes generados en cada cliente, los mensajes de cada llamada son transportados por la red utilizando conexiones periódicas.

En la red existirá un conjunto de llamadas $\{V^1, V^2, ..., V^m\}$. Cada llamada V^k puede ser representada por $(S_s^k, S_e^k, M^k, T^k, D^k)$, donde cada término es definido como:

S_s^k identificación de la estación origen.

S_e^k identificación de la estación destino

M^k número de bits enviado en cada período

T^k período

D^k vencimiento extremo a extremo

Cada enlace de la red, en general, tiene asignadas varias conexiones. Dada una llamada V^k, se notará τ^k_j a la *j*-ésima conexión de la *k-esima* llamada.

Cuando una estación desea establecer una llamada V^k, utiliza los servicios de red especificando las características de dicha llamada en términos de $S_e{}^k$, M^k, T^k y D^k. El algoritmo para determinación de conexiones, selecciona una ruta origen-destino sobre un conjunto de enlaces y asigna un período T^k_j y un vencimiento D^k_j para la conexión en el enlace *j* correspondiente a la llamada V^k.

El procedimiento por el que se establece una conexión es por naturaleza asincrónico. Durante la fase de establecimiento, la red puede rechazar o aceptar una llamada en función de las características temporales de la misma y del estado de la red en ese instante. Se considerará el análisis de la diagramabilidad de una red de conmutación de paquetes en el que existen un conjunto de llamadas de tiempo real ya establecidas y permanentes. Sin embargo, el análisis de diagramabilidad propuesto, puede utilizarse en la fase de establecimiento para analizar la factibilidad de una llamada.

3. Isomorfismo: validación de una ruta tentativa

En 1994, Sathaye ([34]) propuso que el análisis de la diagramabilidad de una llamada establecida entre una estación origen y una destino en una red de conmutación de paquetes multihop, podía considerarse similar a diagramar un trabajo en un sistema multiprocesador. Con pequeñas modificaciones, la noción puede extenderse para plantear formalmente un isomorfismo para el análisis de la diagramabilidad de estos dos sistemas. En la figura 7.1 se define una correspondencia uno-a-uno que establece una

relación de isomorfismo entre sistemas multiprocesador y redes multihop.

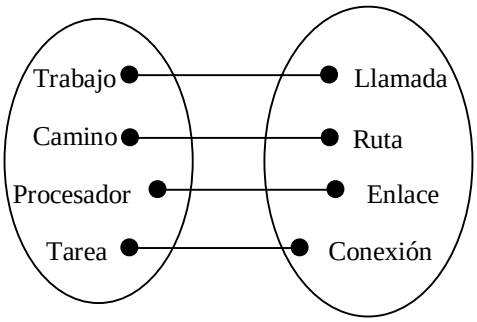


Figura 7.1

Un caso particular de precedencias en sistemas multitarea, es aquél en el cual los trabajos son restringidos a ser una cadena de tareas ejecutadas en diferentes procesadores. En un trabajo del tipo cadena, todas las tareas que lo integran tienen sólo una predecesora y sólo una sucesora, excepto la tarea raíz que no posee predecesora y la tarea hoja que no posee sucesora.

En el isomorfismo propuesto, al trabajo en sistemas multiprocesador le corresponde la llamada en sistemas de conmutación de paquetes multihop. Un camino, el cual es una secuencia de tareas, le corresponde una ruta que es una secuencia de conexiones. A las tareas raíz y hoja le corresponde las conexiones origen y destino respectivamente. Debido a que los trabajos considerados son del tipo cadena, existe un solo camino desde la tarea raíz a una tarea arbitraria del trabajo τ_i, lo que corresponde, en forma isomorfa, a la única ruta que existe entre la conexión origen y una conexión arbitraria τ_j de una llamada.

Al comienzo de una ranura, el procesador es concedido por el diagramador a alguna de las tareas asignadas a él, para su ejecución. Similarmente, al comienzo de cada ranura, el uso del enlace es asignado por el diagramador a una conexión. Como puede observarse, la correspondencia preserva la estructura lógica de las relaciones de los elementos, con lo que los sistemas puede, para el análisis de la diagramabilidad, considerarse *isomorfos* [10].

Un algoritmo de ruteo dinámico puede ser utilizado, en llamadas que

no son de tiempo real, para incrementar la eficiencia en la utilización de la red [2]. Una ruta tentativa es encontrada cuando las restricciones de tiempo real son satisfechas en cada enlace. Esto no garantiza, sin embargo, que el vencimiento extremo a extremo de la llamada es satisfecho. Debido al isomorfismo, el análisis de la diagramabilidad de sistemas multitarea-multiprocesador desarrollado en el capítulo 6 puede ser utilizado para analizar la diagramabilidad de redes de conmutación de paquetes multihop. Para utilizar el desarrollo del capítulo 6 se utilizará la siguiente notación:

τ^j_i, indicará la *i*-ésima conexión de la llamada *j*.

C^j_i, simbolizará el máximo tiempo de utilización del enlace de la conexión τ^j_i. Debido a que el tiempo requerido para transmitir un paquete define el tiempo de ranura, C^j_i coincide con el número de paquetes del mensaje de la llamada V^j. Debe considerarse que, además de los datos transmitidos por la llamada, es necesario considerar la transmisión del contador del mecanismo de generación temporizada.

λ^j_i, simbolizará la ruta de la llamada *j* desde la conexión origen a la conexión τ^j_i.

MTR^j_i, indicará el máximo número de ranuras necesarias para que la conexión τ^j_i pueda transmitir completamente su mensaje.

Con esta notación, la ecuación 6.1 tiene la misma estructura para el análisis de redes de conmutación de paquetes que los sistemas multiprocesador:

$$MTR_i^j = \min t \mid t = C_i^j + \sum_{\forall h,k \mid \tau_h^k \in \mathbf{L}_i^j} C_h^k \cdot \left\lceil \frac{t}{T_h^k} \right\rceil .$$

Sin embargo, debe leerse de diferente manera: la transmisión del mensaje asociado a la conexión τ_i^j finalizará en la C^j_i-ésima ranura libre, que dejan las conexiones pertenecientes a $\mathbf{L}^j_i$ luego de una generación simultánea.

C^j_i es el máximo tiempo necesario para colocar todos los bits del mensaje en el enlace. Luego de colocar todos los bits en el enlace, la señal que codifica el mensaje se propagará por el medio físico a una velocidad medida, generalmente, como fracción de la velocidad de la luz en el vacío. El tiempo necesario para que el mensaje se propague entre dos estaciones unidas por un enlace se denominará: *retardo de propagación*. Δ^j_i simbolizará el retardo de propagación de la conexión τ^j_i, en ranuras.

Λ^j_i en el caso de redes de conmutación de paquetes es un conjunto de un solo elemento puesto que existe una sola ruta desde la conexión origen a cualquier conexión de la llamada. Por lo tanto, $\Lambda^j_i = \{\lambda^j_i\}$.

TEC^j_i simbolizará el máximo tiempo de transmisión y propagación de la ruta λ^j_i. La ecuación 6.2 para el caso de redes de conmutación de paquetes tiene una forma más sencilla:

$$TEC^j_i = \max\left\{ \sum_{\substack{\forall k | \tau^j_k \in \lambda^j_l \\ \tau_h \in \lambda^j_l \wedge \tau_k \prec \tau_h}} \left(MTR^j_k + \Delta^j_{k,h} \right) \right\} \qquad (7.1)$$

El teorema 6.1, que determina la diagramabilidad de una red de conmutación de paquetes multihop, es expresado: una llamada *j* será diagramable si,

$$D_j \geq TEC^j_i \text{ tal que } \tau^j_i \text{es conexión hoja .}$$

Esta expresión debe leerse: La llamada *j* satisfará sus restricciones de tiempo real si su vencimiento extremo a extremo es mayor o igual a la suma de los peores casos de transmisión de las conexiones que la integran, más los retardos de propagación.

Por definición, la red de conmutación de paquetes multihop será diagramable si y sólo si cada una de las llamadas que integran el sistema es diagramable.

Para garantizar la generación periódica de las conexiones de una llamada, se propone el mecanismo de generación temporizada descripto en el capítulo 6 para sistemas multiprocesador. El contador es transmitido al final del mensaje de la llamada, formando parte de éste.

4. Comparación con otros métodos: mecanismo de generación temporizada vs. stop-and-go queues.

En su tesis doctoral [33] y en [34], Sathaye trata el problema de sincronización de las generaciones de las llamadas de una red de conmutación de paquetes multihop en tiempo real. Para resolverlo, utiliza un mecanismo de doble buffer denominado *stop-and-go queuing*, propuesto por S. Golestani en [8]. La idea básica es que los paquetes recibidos en cualquier instante de un período, son generados al comienzo del próximo período de la conexión correspondiente. Para esto, los paquetes recibidos son colocados en la cola de "stop", y son promovidos al comienzo del siguientes período de la conexión a la cola de "go". El diagramador selecciona el paquete que será transmitido por el enlace, inspeccionando la cola de "go" de cada una de las conexiones asignadas a dicho enlace. El diagramador utiliza una disciplina de Períodos Monotónicos Crecientes para la selección de la tarea que utilizará el enlace.

Si los máximos tiempos de ejecución son iguales a los períodos, el peor caso de retardo producido en las colas de una conexión, en un nodo, es el doble del período. Si el máximo tiempo de ejecución y el período son diferentes, el peor caso de retardo es la suma de ambos. Con la notación utilizada en esta tesis, el máximo tiempo de latencia extremo a extremo de una llamada queda expresado de la siguiente forma:

$$TEC^j = n_j . MTR_i^j + \sum_{\forall i | \tau_i^j \in \lambda_i^j} T_i^j \left\lceil \frac{\Delta_i^j}{T_i^j} \right\rceil \qquad (7.2)$$

donde n_j simboliza el número de enlaces entre la estación origen y la

destino, y los *MTR* en cada enlace se consideran iguales. Obviamente, el sistema es diagramable si para cada una de las llamadas $D^j \geq TEC^j$.

La ecuación de diagramabilidad de Sathaye es válida si, además de estar las estaciones del sistema sincronizadas (tienen idéntica duración de la ranura), la conexión sucesora está desfasada en adelanto un MTR con respecto a la conexión predecesora. Esta consideración implica una gran complejidad adicionada al diagramador de cada enlace, pues hace necesario sincronizar cada cola stop-and-go con su desfasaje correspondiente.

4.1 Ejemplo

Para comparar el método de generación temporizada con de colas stop-and-go, se utilizará el ejemplo propuesto por Sathaye en [33]. Se utiliza una estructura del frame de datos del tipo ATM, con de 53 bytes de longitud y con una carga útil de datos (en inglés, payload) de 48 bytes. En una red de 10 Mb/s, el tiempo requerido para transmitir un frame es de 42.4 μs.

Se considerarán cuatro llamadas con diferentes tipos de tráficos:

a) Sonido calidad CD: Muestras de 4 bytes son obtenidas a una frecuencia de 44.1 KMuestras/s. Los frames son ensamblados a períodos de 275 μs o 6 ranuras. Por lo tanto, $C = 1$ y $T = 6$.

b) Voz calidad telefónica: La frecuencia de muestreo es de 8 KMuestras/s, obteniéndose un período de 125 μs. Para esta calidad, 1 byte/muestra es suficiente. Por lo tanto, los frames transportando una carga de 48 bytes son ensamblados con períodos de 6.0 ms ó 141 ranuras. En consecuencia, $C = 1$ y $T = 141$.

c) Imágenes transmitidas a una frecuencia de 1 frame cada 11 ms, equivalente a 259 ranuras. Por lo tanto, $C = 1$ y $T = 259$.

d) Imágenes transmitidas a una frecuencia de 1 frame cada 33 ms, equivalente a 778 ranuras. En consecuencia, $C = 1$ y $T = 778$.

En las figuras 7.2, 7.3, 7.4, 7.5 la latencia extremo a extremo para cada

llamada es ilustrada para diversas distancias extremo a extremo, para 2, 5 y 10 saltos. La distancias entre saltos se consideran iguales. Como puede observarse, cuando los períodos son pequeños, la diferencia entre ambos métodos es pequeña. Sin embargo, cuando el período de la llamada aumenta, el retardo producido por la cola de stop crece en importancia y la latencia extremo a extremo del método de generación temporizada es substancialmente menor que la del mecanismo de colas. Con el incremento del número de saltos extremo a extremo, la ventaja de utilizar el método de generación temporizada se hace más notoria.

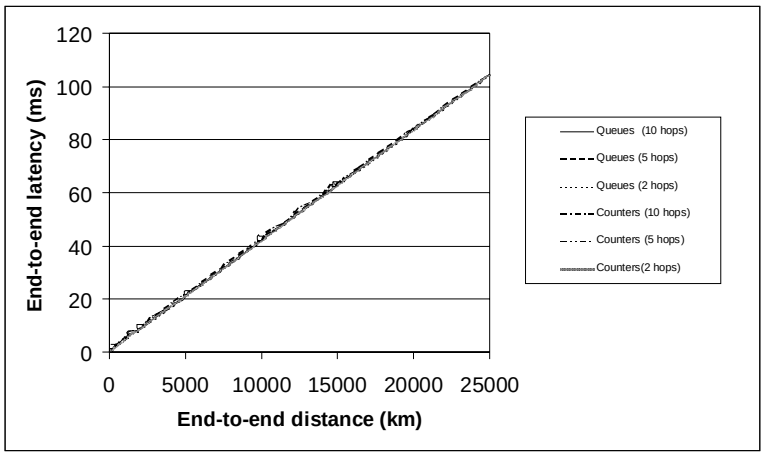


Figura 7.2. Latencia extremo a extremo de llamada de sonido calidad CD de 1 frame cada 275μs

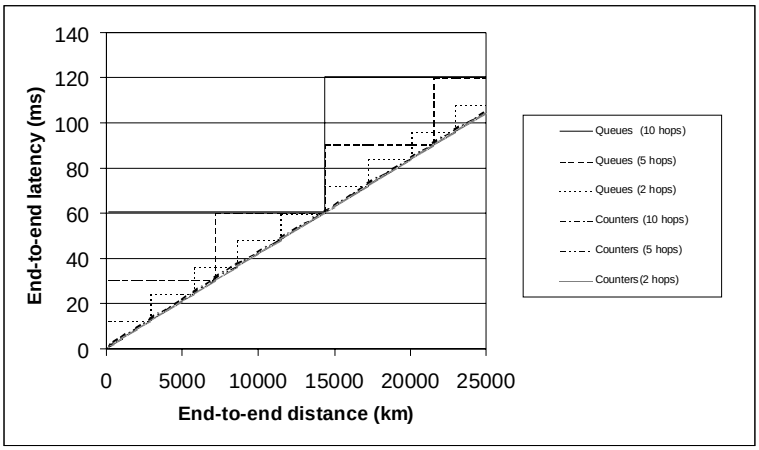


Figura 7.3. Latencia extremo a extremo de llamada de voz telefónica de 1 frame cada 6 ms

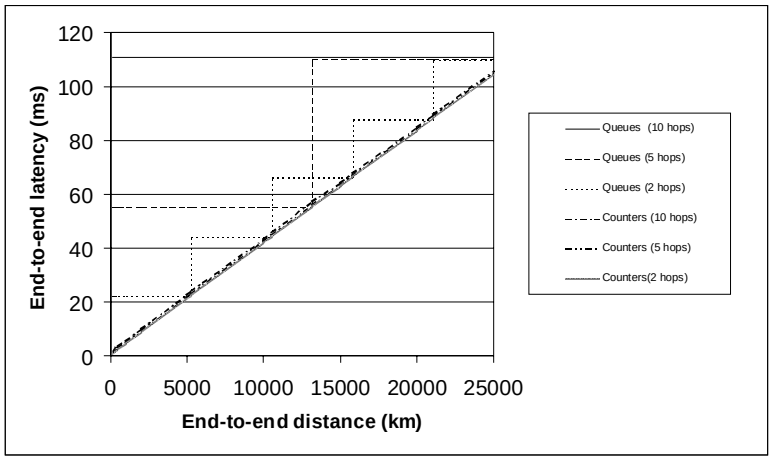


Figura 7.4. Latencia extremo a extremo de llamada de imagen de 1 frame cada 11 ms

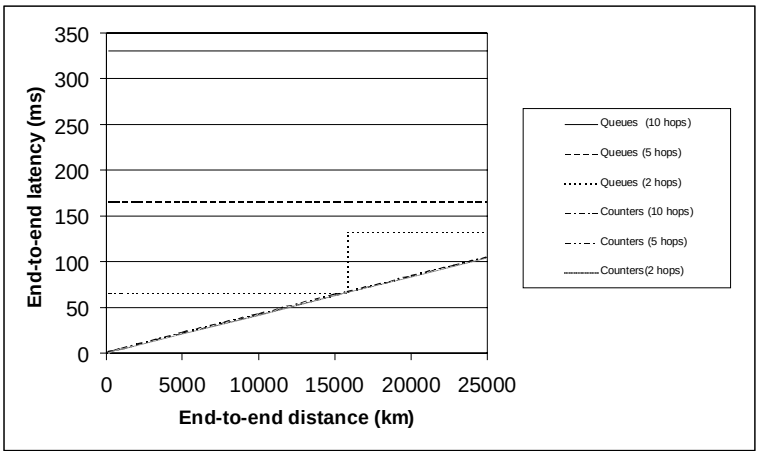


Figura 7.5. Latencia extremo a extremo de llamada de imagen de 1 frame cada 33 ms

Como puede observarse, la curva de latencia del mecanismo de colas no es continua debido a que se producen incrementos igual al número de saltos por el período cada $\left\lceil \frac{\Delta_i^j}{T_i^j} \right\rceil$ ranuras.

Los resultados anteriores pueden ser obtenidos si se analizan las ecuaciones 7.1 y 7.2 aplicándoles las condiciones del ejemplo planteado. Si se substrae de la ecuación 7.2 la latencia extremo a extremo planteada en la ecuación 7.1, la diferencia será el tiempo en exceso que el método de Sathaye tiene sobre el método de generación temporizada. Esta diferencia es:

$$\sum_{\forall i|\tau_i^j \in \lambda_i^j} T_i^j \left\lceil \frac{\Delta_i^j}{T_i^j} \right\rceil - \sum_{\forall i|\tau_i^j \in \lambda_i^j} \Delta_i^j = \sum_{\forall i|\tau_i^j \in \lambda_i^j} T_i^j \left(\frac{\Delta_i^j}{T_i^j} + x \right) - \Delta_i^j = \sum_{\forall i|\tau_i^j \in \lambda_i^j} x \cdot T_i^j \qquad 0 \le x < 1$$

Esto significa que el método propuesto por Sathaye puede adicionar, con respecto al método de generación temporizada, un retardo tan pequeño como cero o tan alto como casi un período de la llamada por cada nodo. Obviamente, períodos mayores y más saltos entre estación origen y destino aumenta la ventaja del método de generación temporizada sobre el método de colas stop-and-go.

Capítulo 8

Conclusiones y Trabajos Futuros

Teniendo en cuenta las características de la disciplina de Prioridades Fijas, Liu y Layland, en 1973, demuestran que el ordenamiento por Períodos Monotónicos Crecientes de la pila de prioridades es óptimo y establecen una condición suficiente para determinar su diagramabilidad.

Recién a fines de la década del 80 fueron publicadas condiciones necesarias y suficientes para determinar la diagramabilidad de sistemas aplicando la disciplina de Períodos Monotónicos Crecientes. Se obtuvieron así condiciones para sistemas de tareas con tiempos de ejecución arbitrarios, bloqueos entre ellas, con un número restringido de prioridades, etc. Sin embargo en todos ellos se supone que las tareas son independientes.

En esta tesis se demostró que en algunos casos, partiendo las tareas independientes en cadenas de subtareas vinculadas por relaciones de precedencias,

se puede mejorar la diagramación hasta el punto de hacer diagramable un sistema no-diagramable por Períodos Monotónicos Crecientes. Futuros trabajos deberán proveer metodologías para establecer las prioridades y generar particiones de tareas en subtareas con relaciones de precedencia. Deberá generalizarse la condición de diagramabilidad para sistemas con restringido número de prioridades y bloqueos entre las tareas.

En lo que respecta a sistemas multiprocesador, los resultados que pueden encontrarse en la bibliografía de tiempo real son menos abundantes e incluso nulos en determinadas áreas. Diferentes alternativas han sido propuestas para resolver el problema de asignación de tareas en un sistema multiprocesador. Se pueden mencionar la utilización de Lógica Difusa [24], Recocido Simulado [41] y métodos heurísticos [26, 30], entre otras, para obtener asignaciones tentativas en sistemas multiprocesador. En el proceso de elaboración de la asignación se va verificando el cumplimiento de todas las constricciones, excepto precedencias que pasan a verificarse recién luego de completar la asignación tentativa. Si son satisfechas, la asignación es validada y se obtuvo una solucion al problema.

Las precedencias consideradas son del tipo de las presentadas en Tindell [41] y denominado precedencias blandas en esta tesis. El considerar precedencias blandas presupone que las tareas que componen el trabajo tienen el mismo período, T_t, que éste. De esta manera, el análisis de la diagramabilidad se reduce a considerar varios sistemas monoprocesador con tareas independientes y considerar el retardo de comunicaciones a través de la red. Sin embargo, lo único que asegura este método es que cada T_t unidades de tiempo se obtendrán resultados de ejecución de sucesivas instancias del mismo trabajo, pero de ninguna manera que cada una de ellas es ejecutada a lo sumo en T_t.

El método de generación temporizada propuesto, permite diagramar trabajos en donde su vencimiento sea menor o igual que su período, para lo cual es necesario utilizar precedencias duras. Futuros trabajos se orientarán a estudiar condiciones de diagramabilidad menos pesimistas y la influencia de la desincronización entre los procesadores en el análisis de la diagramabilidad.

El isomorfismo propuesto entre sistemas multitarea y redes de conmutación

multihop, permite extender el análisis de la diagramabilidad de sistemas multiprocesador a este tipo de redes. El mecanismo de contadores permite temporizar la generación de cada conexión que integra la ruta de una llamada. La condición de diagramabilidad propuesta en [33] es comparada con la propuesta en esta tesis, utilizando el ejemplo planteado en dicho trabajo. Se puede concluir que la condición de diagramabilidad obtenida en esta tesis es menos pesimista que la obtenida por Sathaye [33] y que el método de los contadores es más eficiente para controlar la diagramabilidad de las llamadas que el mecanismo de las colas stop-and-go.

Apéndice A

Símbolos y Definiciones

S(***m***)	Conjunto de *m* tareas de tiempo real.
$\lceil x \rceil$	Operador techo.
τ_i	*i*-ésima tarea de un conjunto de tareas independientes.
T_i	Mínimo tiempo entre generaciones consecutivas de la *i*-ésima tarea o trabajo.
D_i	Vencimiento de la *i*-ésima tarea o trabajo.
C_i	Máximo tiempo de ejecución de la tarea τ_i.
Trabajo	Conjunto de tareas con relaciones de precedencias entre ellas.
PCB	Lugar de memoria donde el sistema operativo almacena toda la información de la tarea (process control block).
PMC	Períodos Monotónicos Crecientes (en inglés rate monotonic).
$\tau_1 \prec \tau_2$	τ_1 precede a τ_2.
O( )	Complejidad.
$\mathbf{J}^h$	Trabajo *h.*
τ_i^j	*i*-ésima tarea del trabajo *j*.
$P(\tau_i)$	Prioridad de la tarea τ_i.
MTR_i	Máximo tiempo de ejecución de la tarea τ_i.

Φ^j_i	Conjunto de tareas predecesoras de τ_i^j.
λ^j_i	Secuencia de tareas que integran, mediante sus relaciones de precedencia, un camino desde una tarea hoja hasta la tarea τ_i^j.
Λ^j_i	Conjunto de todos los caminos posibles hasta la tarea τ_i^j.
MTG^j_i	Máxima cantidad de ranuras que la tarea τ_i^j podrá tener desplazada su generación con respecto a la generación del trabajo.
$\mathbf{L}^j_i$	Conjunto de tareas obtenido de la unión de: - conjunto de tareas que, no perteneciendo al trabajo *j*, tienen una prioridad mayor que τ_i^j. - conjunto de tareas que, perteneciendo al trabajo *j*, tienen mayor prioridad que τ_i^j y deben ejecutarse en el intervalo en donde τ_i^j debe ejecutarse.
TEC^j_i	Tiempo máximo de ejecución de los caminos pertenecientes a Λ^j_i.
UBD	Máximo retardo de comunicaciones (upper bound delay)
RV	Valor del contador que sincroniza la generación de tareas en un sistema multitarea-multiprocesador (release value).
V^k	Llamada *k* en redes de conmutación de paquetes multihop.
S_s^k	Estación origen de la llamada V^k.
S_e^k	Estación destino de la llamada V^k.
M^k	Cantidad de bits enviados por la llamada V^k en cada período.
T^k	Período de la llamada V^k.
D^k	Vencimiento extremo a extremo de la llamada V^k.
MTT^j	Mínimo tiempo de ejecución posible del trabajo *j*

Apéndice B

Lemas y Teoremas

Lemas y Teoremas del Capítulo 4

Teorema 4.1 (Liu and Layland [16])

Si existe un ordenamiento por Prioridades Fijas de la lista de prioridades que haga diagramable el sistema, entonces el ordenamiento por Períodos Monotónicos Crecientes hará diagramable al sistema.

Teorema 4.2 (Liu and Layland [16])

Un conjunto de *m* tareas independientes y periódicas diagramadas mediante la disciplina de períodos monotónicos crecientes será diagramable si:

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq m \cdot \left(2^{1/m} - 1 \right)$$

C_i / T_i es definido en [16] como el *factor de utilización* de la tarea *i*. Este factor representa la fracción de tiempo de procesador utilizado en la ejecución de la tarea *i*.

Teorema 4.3 (Liu and Layland [16])

Dado un conjunto de *m* tareas independientes, **S**(*m*), éste será diagramable por la disciplina de menor tiempo al vencimiento si y sólo si:

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq 1$$

Teorema 4.4 (Mok [18])

La disciplina de menor tiempo al vencimiento no es óptima para sistemas multitarea-multiprocesador.

Teorema 4.5 (Mok [18])

Para un sistema con dos o más procesadores, ningún algoritmo basado en vencimientos puede ser óptimo sin un completo conocimiento *a priori* de los vencimientos, máximos tiempos de ejecución e instantes de arribos.

Anomalía de Richard [8]

Dado un conjunto de tareas óptimamente diagramado con alguna disciplina de prioridades sobre un conjunto fijo de procesadores, tiempos de ejecución fijos y relaciones de precedencia, entonces el sistema puede dejar de ser diagramable si: se modifica la pila de prioridades, se incrementa el número de procesadores, se reduce el tiempo de ejecución de las tareas o se alteran las relaciones de precedencia.

Lemas y Teoremas del Capítulo 5

Lema 5.1

Una tarea τ_i, raíz de un trabajo con período *T*, puede tener un máximo de $\lceil t / T \rceil$ generaciones en un intervalo de *t* ranuras.

Lema 5.2

Una tarea τ_j, sucesora de una tarea τ_i, y perteneciente a un trabajo con período *T*, puede tener un máximo de $\left\lceil t/T \right\rceil$+1 generaciones en un intervalo de *t* ranuras.

Lema 5.3

Si τ_i es tarea raíz y τ_k es una tarea perteneciente a otro trabajo cuya predecesora tiene menor prioridad que τ_i, entonces podrá existir como máximo una ejecución completa de τ_k y sus sucesivas sucesoras con mayor prioridad que τ_i en el intervalo definido por la generación de τ_i y su ejecución completa.

Lema 5.4

El máximo tiempo de ejecución de una tarea raíz τ_i, MTR_i, será:

$$MTR_i = \min t | t \geq C_i + \sum_{\forall \tau_h \in \mathbf{R}_i} C_h \left\lceil \frac{t}{T_h} \right\rceil + \sum_{\forall \tau_h \in \mathbf{T}_i} C_h \left(\left\lceil \frac{t}{T_h} \right\rceil + 1 \right) + \sum_{\forall \tau_h \in \mathbf{U}_i} C_h$$

donde

$$\mathbf{R}_i = \left\{ \tau_h : \nexists \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_i) \right\}$$

$\mathbf{T}_i = \mathbf{A}_i^r | \mathbf{A}_i^r = \mathbf{A}_i^{r+1}$ y $\nexists\, q < r$ tal que $\mathbf{A}_i^r = \mathbf{A}_i^q$, en donde $\mathbf{A}_i^k$ es definido en forma recursiva como

$$\mathbf{A}_i^0 = \left\{ \tau_h : \exists \tau_p \in \mathbf{R}_i | \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_i) \right\}$$

$$\mathbf{A}_i^k = \left\{ \tau_h : \exists \tau_p \in \mathbf{A}_i^{k-1} | \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_i) \right\} \cup \bigcup_{l=0}^{k-1} \mathbf{A}_i^l$$

$\mathbf{U}_i = \mathbf{B}_i^r | \mathbf{B}_i^r = \mathbf{B}_i^{r+1}$ y $\nexists\, q < r$ tal que $\mathbf{B}_i^r = \mathbf{B}_i^q$, en donde $\mathbf{B}_i^k$ es definido en forma recursiva como

$$\mathbf{B}_i^0 = \left\{ \tau_h : \exists \tau_p \prec \tau_h | \left(P(\tau_p) < P(\tau_i) \right) \wedge \left(P(\tau_h) > P(\tau_i) \right) \right\}$$

$$\mathbf{B}_i^k = \left\{ \tau_h : \exists \tau_p \in \mathbf{B}_i^{k-1} | \left(\tau_p \prec \tau_h \right) \wedge \left(P(\tau_h) > P(\tau_i) \right) \right\} \cup \bigcup_{l=0}^{k-1} \mathbf{B}_i^l$$

Lema 5.5

No pueden existir simultáneamente requerimientos pendientes de dos sucesoras τ_j y τ_l, pertenecientes a diferentes trabajos, cuyas

correspondientes predecesoras, τ_i y τ_k, tienen menor prioridad que la menor prioridad de ellas. Es decir, τ_j y τ_l no podrán tener requerimientos pendientes simultáneamente si máx($P(\tau_i)$,$P(\tau_k)$) < mín($P(\tau_j)$,$P(\tau_l)$)

Lema 5.6

El máximo tiempo de ejecución de una tarea τ_j, MTR_j, sucesora de τ_i con $P(\tau_i)<P(\tau_j)$, será:

$$MTR_j = \min t | t \geq C_j + \sum_{\forall \tau_h \in \mathbf{R}_j} C_h \left\lceil \frac{t}{T_h} \right\rceil + \sum_{\forall \tau_h \in \mathbf{T}_j} C_h \left(\left\lceil \frac{t}{T_h} \right\rceil + 1 \right)$$

donde

$$\mathbf{R}_j = \left\{ \tau_h : \exists \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_j) \right\}$$

$\mathbf{T}_j = \mathbf{A}_j^r | \mathbf{A}_j^r = \mathbf{A}_j^{r+1}$ y $\nexists\ q < r$ tal que $\mathbf{A}_j^r = \mathbf{A}_j^q$, en donde $\mathbf{A}_j^k$ es definido en forma recursiva como

$$\mathbf{A}_j^0 = \left\{ \tau_h : \exists \tau_p \in \mathbf{R}_j | \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_j) \right\}$$

$$\mathbf{A}_j^k = \left\{ \tau_h : \exists \tau_p \in \mathbf{A}_j^{k-1} | \tau_p \prec \tau_h \wedge P(\tau_h) > P(\tau_j) \right\} \cup \bigcup_{l=0}^{k-1} \mathbf{A}_j^l$$

Teorema 5.1

Un trabajo será diagramable, si la mayor sumatoria de los *MTR* de las tareas que integran las diferentes secuencias, predecesora-sucesora de un tarea raíz a una tarea hoja, es menor que el vencimiento de dicho trabajo.

Corolario

Un sistema será diagramable si todos los trabajos que lo integran son diagramables.

Lemas y Teoremas del Capítulo 6

Lema 6.1

El MTR^j_i estará dado por:

$$MTR_i^j = \min t \mid t = C_i^j + \sum_{\forall h,k \mid \tau_h^k \in \mathbf{L}_i^j} C_h^k \cdot \left\lceil \frac{t}{T_h^k} \right\rceil$$

Lema 6.2

El máximo tiempo de generación de la tarea τ^j_i, MTG^j_i, será:

$$MTG_i^j = \max\left\{ \sum_{\forall h,k \mid \tau_k^j \in \lambda_l^j \wedge \tau_k^j \prec \tau_h^j} \left(MTR_k^j + \Delta_{k,h}^j \right) \right\} \ \forall\ \lambda_l^j, \Lambda_l^j, \tau_l^j \mid \tau_l^j \in \Phi_i^j$$

Teorema 6.1

El trabajo *j* será diagramable si para toda tarea hoja τ^j_i se cumple:

$D_j \geq \max \{\ TEC_i^j\ \}\ \forall \tau_i^j \mid \tau_i^j hoja$

Corolario

El sistema será factible con el diagramador propuesto si para todos los trabajos del sistema se cumple el Teorema 6.1.

Lema 6.3

El menor MTR^j_i que podrá tener la tarea τ^j_i es su máximo tiempo de ejecución C^j_i.

Teorema 6.2

El mínimo tiempo de ejecución que puede tener el trabajo *j*, simbolizado MTT^j, en su peor caso de ejecución, estará determinado por el máximo TEC^j_i de todas las tareas τ^j_i hojas de dicho trabajo, considerando $MTR^j_i = C^j_i$.

$$MTT^j = \max \sum_{\forall \tau_k \in \lambda_i^j} C_k^j \qquad \forall \tau^j_i \text{ tarea hoja}$$

Corolario

El trabajo *j* será intrínsecamente no diagramable si $D^j > MTT^j$.

Teorema 6.4

Si para todos los trabajos a asignar $MTT^j \leq D^j$, entonces existirá un conjunto de *x* procesadores para el cual el sistema será diagramable.

Referencias

[1] Audsley, N.C., Burns, A., Richardson, M. F., and Wellings, A. J. "Hard Real-Time Scheduling: The Deadline Monotonic Approach". *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, GA, USA, pp. 127-132. 1991.

[2] Ash, G. "Dynamic network evolution, with examples from AT&T's evolving dynamic network", *IEEE Communications*, **33**(7): pp. 26-39. 1995.

[3] Baker, T.P., and Shaw, A. "The cyclic Executive Model and Ada". *Real-Time Systems Journal*, **1**(1): pp. 7-25. 1989.

[4] Bianchini, R. "Interprocessor traffic scheduling and multicommodity network flows", *Ph. D. Thesis*, Carnegie Mellon University, Pittsburgh, PA, March 1989.

[5] Cayssials, R., Orozco, J., Santos, J., and Santos, R., "Rate Monotonic scheduling of real-time control systems with the minimum number of priority levels", *Proc. 11th Euromicro Workshop on Real Time Systems '99*, IEEE Computer Society Press, York, England, 1999.

[6] Cayssials, R., Santos, J., Orozco, J., and Ferro, E., "On the Scheduling of Real-Time Multihop Packet-Switching Networks", *Proc. 10th Euromicro Workshop on Real Time Systems '98*, IEEE Computer Society Press, Berlin, 1998.

[7] Chetto, H., Silly, M., and Bouchentouf, T. "Dynamic Scheduling of Real-Time Task under Precedence Constraints". *Real-Time Systems Journal*, **2**(2):pp. 181-194. 1990.

[8] Golestani, S. "A stop-and-go queuing framework for congestion management", *Proc. SIGCOM'90*, June 1990.

[9] Graham, R., "Bounds on the Perfomance of Scheduling Algorithms", *Computer and Job Shop Scheduling Theory*, E.G. Coffman, ed., Jhon Wiley and Sons, pp. 165-227. 1976.

[10] Harrison, M., "Introduction to Switching and Automata Theory", McGraw-Hill. 1965.

[11] Joseph, M., and Pandya, P. "Finding Response Times in a Real-Time System", *The Computer Journal*, **29**(5):pp. 390-395. 1986.

[12] Katcher, D.I., Arakawa, H., and Strosnider, J.K. "Engineering and Analysis of Fixed Priority Schedulers". *Transactions on Software Engineering* **19**(9), pp. 920-934, Sep. 1993.

[13] Kieckhafer, R.M. "Fault-Tolerance Real-Time Task Scheduling in the MAFT Distributed System". *22nd Hawaii International Conference on System Sciences*, Jan. 1989.

[14] Lawler. "Optimal Sequencing of a Single Machine Subject to Precedence Constraints", *Management Science*, N° 19, 1973.

[15] Lehozcky, J.P., Sha, L., and Ding, Y. "The Rate Monotonic Scheduling Algorithm: Exact characterization and average case behavior", *Proc. Real Time Symp. IEEE CS*, Los Alamitos, CA. 1989.

[16] Lenstra, J.K., and Rinnooy Kan A.H.G. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey", *Ann. Discrete Math*, N° 5, pp. 287-326. 1977.

[17] Leung, J., and Whitehead, J. "On the complexity of Fixed Priority Sheduling of periodic, Real -Time tasks", *Perfomance Evaluation*, **2**(4), pág. 237-250, 1982.

[18] Lim, M. "Implementing fuzzy rule-based systems on silicon chips", *IEEE Expert*, **5**(1): pp. 31-45. 1990.

[19] Liu, C.L., and Layland, J.W. "Scheduling algorithms for multiprograming in hard real time enviroments", *J. ACM*, **20**(1): pp. 46-61. 1973.

[20] Lynx Programmer's Referenca Manual, Version 2.4, *Lynx Real-Time Systems*, San Jose, CA. 1996.

[21] Mok, A.K., "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment", PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, Mass. May 1983.

[22] Obenza, R. "Rate Monotonic analysis for real-time systems". *IEEE Comput*. **26**(3): pp. 73-73. 1993.

[23] Orozco, J., Cayssials, R., Santos, J., and Ferro, E., "Precedence Constraints in Hard Real Time Distributed Systems", *Proc. Third IEEE International Conference on Engineering of Complex Computer Systems*, IEEE Computer Society Press, Italy, 1997

[24] Orozco, J., Cayssials, R., Santos J., and E. Ferro, "Design of a Learning Fuzzy Production System to Solve an NP-Hard Real-Time Assignment Problem", *Proc. 8-th Euromicro Workshop on Real-Time Systems*, L'Aquila, Italy, June 12-14, pp. 146-150. 1996.

[25] Perng-Yi M.R., Lee E.Y.S., and Tsuchiya, M. "A Task Allocation Model for Distributed Computing Systems". *IEEE Transactions on Computers*, **31**(1): pp. 41-47. 1982.

[26] Ramamritham, K., "Allocation and scheduling of complex periodic task", *In Proc. 10th International Conference on Distributed Computing Systems*, pp. 108-115, 1990.

[27] Ramamritham K., Stankovic J., and Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessors Systems", *IEEE Trans. Parallel and Distr. Computting*, **4**(4): pp. 382-397. 1993.

[28] Ramamrithan, K., and Stankovic, J. "Scheduling Algorithms and Operating Systems Supports for Real-Time Systems". *Proceedings of the IEEE*, **82**(1): pp. 55-67, Jan. 1994.

[29] Santos J., and Ferro E., "An Heuristic Aproach to Allocating and Scheduling NP-Hard Real-Time Systems", *XX Conferencia latinoamericana de Informática*, pp. 739-746. 1994.

[30] Santos, J., Ferro, E., Orozco, J., and Cayssials, R. "A Heuristic Approach to the Multitask-Multiprocessor Assignment Problem using the Empty-Slots Method and Rate Monotonic Scheduling". *Real-Time Systems Journal*. Kluwer Academic Publishers, **13**, 167-199. 1997.

[31] Santos, J., and Orozco, J., “Rate Monotonic Scheduling in Hard Real-Time Systems”, *Information Processing Letters*, **48**, pp. 39-45. 1993.

[32] Santos J., Orozco J., and Alimenti O., “Perfomance Evaluation of Standard Lan Protocols in Time Constrained enviroments”, *IEEE INFOCOM´89*. Waterloo, Ontario, Canadá, 1989.

[33] Sathaye, S.S. "Scheduling Real-Time Traffic in Packet-Switched Networks", *PhD Thesis*, Carnegie Mellon University, Pittsburgh, PA, June 1993.

[34] Sathaye, S.S, and Strosnider, J.K. "A Real-Time Scheduling Framework for Packet-Switched Networks", *14th International Conference on Distributed Computing Systems*. May 1994.

[35] Sha L., Rajkumar R., and Lehoczky J.P., “Priority Inheritance Protocols: An aproach to Real-Time Synchronization”, *IEEE Trans. Computers*, **39**(9): pp. 1175-1185. 1990.

[36] Silberschatz, A., and Galvin, P. "Operanting Systems Concepts". *Addison Wesley*, Fourth Edition, 1994.

[37] Stankovic, J.A. "Misconceptions About Real-Time Computing". *IEEE Computer*, **21**(10): pp. 10-19, October 1988.

[38] Stankovic, J.A. "Real-Time Computing Systems: The Next Generation". *Hard Real-Time Systems by John A. Stankovic and Krithi Ramamritham, IEEE Computer Society Press*, pp. 14-37. 1988.

[39] Stankovic, J.A., and Ramamritham, K. "The Spring Kernel: A New Paradigm for Real-Time Systems". *IEEE Software*, **8**(3): pp. 62-72. May 1991.

[40] Stankovic J.A., Spuri M., Di Natale M., Buttazzo G.C. “Implications of Classical Scheduling Results for Real-Time Systems”, *IEEE Computer* , **28**(6): pp 16-25. 1995.

[41] Tindell, K., Burns, A., and Wellings, A. “Allocating Hard Real-Time tasks: An NP-hard problem made easy”, *Real Time Systems,* **4**:pp. 145-166. 1992.

[42] Tokuda, H., Nakajima, T., and Rao, P. "Real_time Mach: Toward a Predictable Real-Time System". *Proceedings of USENIX Mach Workshop*, Oct. 1990.

[43] Wilfong, G.T. "Guest Editor's Introduction: Robotics and Automation", *IEEE Computer*, **22**(3); pp. 6-7, Mar 1989.

[44] Xu, J., and Parnas, D.L. "Scheduling processes with release times, deadlines, precedence and exclusion relations", *IEEE Trans. Software Eng*, **16**(3): pp. 360-369. 1993.

[45] Zhao W., Ramamritham K., Stankovic J., “Preemptive Scheduling under Time and Resource Constraints”, special issue on *Real-Time Systems, IEEE Trans. Computers*, **36**(8): pp. 949-960. 1987.