C.L. Liu, James W. Layland. Jet Propulsion Laboratory, California Institute of Technology

Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

Una computadora de control de proceso realiza una o más funciones de control y monitoreo. Apuntar con una antena y registrar si una nave espacial está en orbita is una de los ejemplos de dichas funciones. Cada función que se realiza está asociada con un conjunto de una o más tareas. Algunas de estas tareas son ejecutadas en respuesta a eventos en un equipo que es controlado o monitoreado por una computadora. Las restantes son ejecutadas en respuesta a eventos de otras tareas. Ninguna de estas tareas puede ser ejecutadas antes que el evento que se solicitado ocurra. Cada una de las tareas deben ser completadas antes que un tiempo fijo hay transcurrido luego de la solicitud que se les hizo. Servicio dentro de este rango de tiempo debe ser garantizado, lo que cataloga el ambiente como *hard-real-time* en contraste con *soft-real-time* en donde una distribución estadística de los tiempos de respuesta es aceptable.

The Environment. Para obtener algun resultado analítico acerca del comportamiento del programa en un ambiente *hard-real-time*, ciertas suposiciones deben ser hechas acerca del ambiente. No todas estas suposiciones son absolutamente necesarias: **(A1)** Las solicitudes para todas las tareas para las cuales existen *hard deadlines* son periódicas, con un intervalo constante entre las solicitudes. **(A2)** Los *Deadlines* consisten solamente de restricciones en la capacidad de ejecución – por ejemplo, cada tarea tiene que ser completada antes que la nueva solicitud para ella ocurra. **(A3)** Las tareas son independientes en el sentido de que las solicitudes para una determinada tarea no dependen de la iniciación o la finalización de solicitudes para otras tareas. **(A4)** El tiempo de ejecución para cada tarea es constante para esa tarea y no varía con el tiempo. Aquí, el tiempo de ejecución se refiere al tiempo que toma al procesador ejecutar una tarea sin interrupción. **(A5)** Cualquier tarea no periódica en el sistema es especial; son rutinas de inicialización o recuperación de fallos; desplazan a las tareas periódicas mientras ellas mismas corren y no tienen *dealines* fuertes(*hard*). Un algoritmo de *scheduling* es un conjunto de reglas que determinan la tarea a ser ejecutada en un momento en particular. Los algoritmos del artículo son apropiativos y basados en prioridades. Un algoritmo de *scheduling* se dice que es **estático** (o algoritmo de *scheduling* de prioridades fijas.) si las prioridades son asignadas a las tareas una vez y no cambian. Un algoritmo de *scheduling* se dice **dinámico** si las prioridades de las tareas pueden cambiar de solicitud en solicitud. Un algoritmo de *scheduling* se dice *mixed scheduling algorithm* si las prioridades de algunas de las tareas son fijas mientras que las prioridades restantes varían de solicitud en solicitud.

A Fixed Priority Scheduling Algorithm. El *deadline* de una solicitud para una tarea se define como el tiempo de la siguiente solicitud para la misma tarea. Para un conjunto de tareas planificadas por un algún algoritmo de *scheduling*, se deice que un *overflow* ocurre en el tiempo t si t es el *deadline* de una solicitud que no

se cumplió. Para un dado conjunto de tareas, un algoritmo de *scheduling* es factible si las tareas son planificadas de tal forma que no ocurra ningún *overflow*. El tiempo de respuesta de una solicitud para cierta tarea se define como el tiempo que hay entre la solicitud y el final de la respuesta para esta solicitud. Un *critical instant* para una tarea se define como un instante en donde una solucitud para dicha tarea tendrá el tiempo de respuesta más largo. Una *critical time zone* para una tarea es el intervalo entre un *critical instant* y el fin de la respuesta de la solicitud correspondiente de una tarea. Teorema 1: Un *critical-instant* para cualquier tarea ocurre cuando la tarea es solicitada simultáneamente con solicitudes para todas las tareas de alta prioridad. Teorema 2: Si una asignación de prioridad factible existe para algún conjunto de tareas, la asignación de prioridad *rate-monotonic* es factible para ese conjunto de tareas.

Achievable Processor Utilization. El factor de utilización del procesador se define como la fracción del tiempo de procesador gastado en la ejecución del conjunto de tareas. En otras palabras, el factor de utilizacio4n es igual a uno menos la fracción de tiempo de procesador inactivo. Dado que C_i/T_i es la fracción de tiempo de procesador gastado en ejecutar la tarea τ, para m el factor de utilización es:

$$U = \sum_{i=1}^{m} (C_i/T_i)$$

. Aunque el factor de utilización puede ser mejorado incrementando los valores de los C_i's o decrementando los valores de los T_i's es limitado superiormente por los requerimientos que todas las tareas satisfagan sus *deadlines* en sus *critical instants*. Correspondiente a una asignación de prioridad, un conjunto de tareas se dice que totalmente utilizable por el procesador si la asignación de prioridad es factible para el conjunto y si un incremento en el tiempo de ejecución de cualquiera de las tareas en el conjunto harán que la asignación de prioridades se vuelva no factible. Para un algoritmo de *scheduling* de prioridades fijas, el *least upper bound* del factor de utilización es el mínimo de los factores de utilizacio4n de todos los conjuntos de tareas que utilizan totalmente el procesador. Para todas los conjuntos de tareas cuyo factor de utilización de procesador está por debajo de este límite, existe una asignación de prioridad fija que es factible. Debido que la asignación de prioridades *rate monotonic* es óptima, el factor de utilización logrado por la asignación de prioridad *rate-monotonic* para un conjunto de tareas es mayor o igual que el factor de utilización de cualquier otra asignación de prioridad para ese conjunto de tareas. Así, el *least upper bound* a ser determinado es el ínfimo del factor de utilización correspondiente a la asignación de prioridad *rate-monotonic* de todas los periódos de solicitudes posibles y de los tiempos de ejecución de las tareas. Teorema 3: Para un conjunto de dos tareas con una asignación fija de prioridades, el *least upper bound* del factor de utilización del procesador es $U = 2(2^{\frac{1}{2}} - 1)$. Teorema 4: Para un conjunto de m tareas con un orden de prioridad fijo, y la restricción que la proporción entre dos periódos de solicitud es menor que 2, el *least upper bound* del factor de utilización del procesador es $U = m(2^{\frac{1}{m}} - 1)$. Teorema 5: Para un conjunto de m tareas con un orden de prioridad fijo, el *least upper bound* de utilización de procesador es $U = m(2^{\frac{1}{m}} - 1)$.

Relaxing the Utilization Bound. En la sección anterior se muestra que el *least upper bound* impuesto sobre la utilización del procesador por el requerimiento de servicio *hard-time* garantizado puede acercarse a $\ln(2)$ para conjuntos grades de tareas. Es deseable encontrar forma de mejorar esta situación, debido a que los costos prácticos de cambiar entre tareas debe ser contado. Una de las formas más simples de hacer el costo del límite de utilización igual a 1 es hacer $\{T_m/T_i\} = 0$ para $i = 1, 2, \ldots, m-1$. Dado que esto no se puede hacer siempre , una solución alternativa es poner en búfer la tarea τ_m y tal vez varias de las tareas de baja prioridad y relajar sus *hard deadlines*.

The Deadline Driven Scheduling Algorithm. Usando este algoritmo, las prioridades son asignadas a las tareas de acuerdo a los *deadlines* de sus solicitudes actuales. A una tarea se le asignará la prioridad más alta si el *deadline* de su solicitud actual es el más cercano, y se le asignará la prioridad más baja si el *deadline* de sus

solicitud actual es el más lejano. En cualquier instante, la tarea con la prioridad más alta y con una solicitud aún no cumplida será ejecutada. Tal método de asignación de prioridades de las tareas un dinámico, en contraste con la asignación estática en donde las prioriedades de las tareas no cambian con el tiempo. Teorema 6: Cuando el algoritmo de *deadline driven scheduling* es usado para planificar un conjunto de tareas en un procesador, no hay tiempo de procesador inactivo antes de un *overflow*. Teorema 7: Dado un conjunto de m tareas, el algoritmo de *deadline driven scheduling* es factible si y solo si

$$(C_1/T_1) + (C_2/T_2) + \cdots + (C_m/T_m) \leq 1$$

A Mixed Scheduling Algorithm. Combinación de algoritmos de *scheduling rate-monotonic* y de *deadline driven*. A esta clase se les llama algoritmos de *scheduling* mixtos. El estudio de estos algoritmos está motivada por la observación que el hardware de interrupciones de las computadores de ese momento actuaban como *schedulers* de prioridad fija y no parecen ser compatibles con el *scheduler* de hardware dinámico. Por otro lado, el costo de implementar un *scheduler* en software para las tareas de ritmo más lento no incrementa significativamente si estas tareas son *deadline driven* en lugar de tener una asignación de prioridad fija. Para ser específico, hacer las tareas $1, 2, \ldots, k$ las k tareas de periódos más cortos a ser planificadas/programadas de acuerdo a un algoritmo de *scheduling* de prioridad fija *rate-monotonic*, y dejar que las tareas restantes, las tareas $k+1, k = 2, \ldots, m$ sean planificadas/programadas de acuerdo a un algoritmo de *scheduling deadline driven* cuando el procesador no está ocupado por las tareas $1, 2, \ldots, k$.

Sea $a(t)$ una función no decreciente de t. Se dice que $a(i)$ es sublinear si para todo t y todo T

$$a(T) \leq a(t+T) - a(t)$$

La función de disponibilidad de un procesador para un conjunto de tareas se define como el tiempo de procesador acumulado desde 0 a t disponbile para este conjunto de tareas. Suponer que k tareas han sido planificadas/programadas en un procesador por un algoritmo de *scheduling* de prioridad fija. Denotar con $a_k(t)$ la función de disponibilidad del procesador para las tareas $k+1, k+2, \ldots, m$. Claramente, $a_k(t)$ es una función no decreciente de t. Además, $a_k(t)$ puede ser vista como sublinear de acuerdo a: Teorema 8: Si un conjunto de tareas están planificadas/programadas por el algoritmo de *deadline driven scheduling* en un procesador cuya función de disponibilidad es sublinear, entonces no hay un periódo de tiempo de procesador inactivo para un *overflow*. Teorema 9: Una condición necesaria y suficiente para la factibilidad del algoritmo de *deadline driven scheduling* con respecto a un procesador con función de disponibilidad $a_k(t)$ es:

$$\lfloor t/T_{k+1} \rfloor C_{k+1} + \lfloor t/T_{k+2} \rfloor C_{k+2} + \cdots + \lfloor t/T_m \rfloor C_m \leq a_k(t)$$

para toda t que es múltiplos de T_{k+1}, o $T_{k+2}, \ldots,$ o T_m.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

El problema de *multiprogram scheduling* es un solo procesador desde el punto de vista de las características propias de las funciones del programa que necesitan garantizar servicio.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

El uso de computadoras para control y monitoreo de procesos industriales se ha expandido mucho en años recientes, y probablemente se va a expandir aún más en el futuro. Usualmente, la computadora usada en tales aplicaciones es compartida entre un número de funciones de tiempo crítico y monitoreo y flujos de procesamiento por lotes no críticos en tiempo. Sin embargo, en otras instalaciones, no ningún trabajo **no** crítico en tiempo existe, y el uso eficiente de la computadora puede ser logrado solamente por medio de una planificación cuidadosa de las funciones críticas en tiempo y monitoreo. El último grupo se podría llamar *"pure process control"* y provee la referencia para los análisis en *combinatoric scheduling* presentados en el artículo.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

Mucha de la literatura disponible en multiprogramación tiene que ver con el análisis estadístico de sistemas comerciales de tiempo compartido. Otro subconjunto tiene que ver con aspectos más interesantes de planificación de una instalación de procesamiento por lotes o una instalación mixta de procesamiento por lotes en tiempo compartido, usualmente bajo una configuración multiprocesador. Manacher, deriva un algoritmo para la generación de planificación de tareas en un ambiente *hard real-time*, pero sus resultados están restringuidos a una situación no realista en las que las tareas tienen solamente una solicitud a la vez, incluso cuando múltiples *deadlines* son considerados. Lampson discute el problema de software *scheduling* en términos generales y presenta un conjunto de procedimientos de multiprogramación en Algol que podrían ser implementadas o diseñadas dentro de un *scheduler* de uso general. El texto de Martin, representa el rango de sistemas que son considerados de "tiempo real" y discute los problemas que se encuentran al intentar programarlos.

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

Dos algoritmos de *scheduling* de tipo *"pure process control"* se estudian. Ambos son orientados a prioridad y apropiamiento, lo que significa que el procesamiento de cualquier tarea es interrumpida por una solicitud de mayor prioridad. El primer algoritmo estudiado usa una asignación de prioridad fija y puede lograr utilización de procesador en el orden del 70 % o más. El segundo algoritmo de *scheduling* puede lograr utilización total del procesador por medio de una asignación dinámica de prioridades. También se presenta una combinación de los dos algoritmos.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

Un algoritmo de *scheduling* que asigna prioridades a tareas en un relación monotónica con us tasas de solicitud se ha mostrado como óptimo entre la clase de todos los algoritmos de *scheduling* de asignación fija. El *least upper bound* del factor de utilización de procesador para este algoritmo está en el orden del 70 % para grandes conjuntos de tareas. El algoritmo de *deadline* dinámico mostró ser globalmente óptimo y capaz de lograr utlización total de procesador. Una combinación de los dos algoritmos de *scheduling* provee la mayoría de los beneficios de los algoritmos orientados a *deadline*.