Sistemas operativos en tiempo real

Miguel Angel Astor Romero

Versión 2 - 21 de septiembre de 2016

Introducción

Aparte de la computación de propósito general tan importante para la sociedad moderna, existe un enorme mercado de computadoras de propósitos específicos que, a pesar de pasar desapercibido para el usuario común, representa la inmensa mayoría de los dispositivos computacionales en uso hoy en día [1]. Un tipo particular de estos dispositivos se conocen como sistemas de tiempo real y se caracterizan, como su nombre sugiere, por ser altamente dependientes en el correcto manejo del tiempo, especialmente al ser comparados con sistemas de propósito general. Este tipo de sistemas son ampliamente utilizados en la robótica, la industria manufacturera, la industria de la salud e incluso en entornos militares; ámbitos caracterizados por necesitar de sistemas robustos capaces de responder de forma oportuna dentro de plazos de tiempo que pueden ser sumamente estrictos.

El resto de este documento se estructura de la siguiente manera. En la Sección 1 se presenta una introducción a los sistemas de tiempo real y a sus sistemas operativos. En la Sección 2 se estudian a fondo las características deseables que debe poseer todo sistema operativo de tiempo real. En la Sección 3 se muestran tópicos a tomar en cuenta al momento de diseñar un sistema operativo de tiempo real. En la Sección 4 se realiza un estudio de distintos métodos para la planificación de procesos en un sistema operativo de tiempo real. Finalmente se presentan las conclusiones.

1. Definición y categorías

De forma general, un sistema de tiempo real es un sistema en el cual el tiempo tiene un rol esencial [2]. Estos sistemas deben ser capaces de reaccionar ante estímulos externos dentro de plazos de tiempo conocidos, siendo tan grave el producir respuestas después del vencimiento de un plazo como el no producir respuesta en absoluto [3]. En otras palabras, el objetivo de un sistema de tiempo real es el de producir respuestas en tiempos conocidos y predecibles [4]. Ejemplos de sistemas de tiempo real son el controlador de un robot en una linea de ensamblado, el piloto automático de un avión comercial o incluso un dispositivo casero para reproducción multimedia como un lector de DVD (*Digital Versatile Disk* - Disco Digital Versátil) [2].

El campo de desarrollo de los sistemas de tiempo real se solapa considerablemente con el de los sistemas embebidos [4]. Sin embargo, es igualmente común el desarrollar sistemas embebidos sin capacidades de tiempo real [5]. En base a esto podemos enunciar las siguientes como las características usuales de un sistema de tiempo real [4]:

1. Son sistemas de un único propósito.
2. Son sistemas de tamaño pequeño.
3. Son de bajo costo y se producen en masa.
4. Tienen requisitos de temporización específicos.

Para poder gestionar y utilizar un sistema de tiempo real es necesario un sistema operativo diseñado específicamente para tal fin. Estos sistemas operativos pueden estar desarrollados a la medida para cierto hardware específico (microcontroladores o microprocesadores) o pueden ser capaces de ser compilados y ejecutados en múltiples plataformas de hardware. Ejemplos de esto último son los sistemas operativos VxWorks [6], RTLinux y EmbeddedLinux [4, 1].

Los sistemas de tiempo real se agrupan en dos grandes categorías según la tolerancia que tengan a los retrasos en el cumplimiento de los plazos de tiempo asignados al sistema. La primera de estas categorías es la de los sistemas de tiempo real suave o débil y la segunda la de los sistemas de tiempo real fuerte o estricto [3].

1.1. Tiempo real suave

Un sistema se considera de tiempo real suave cuando es permisible el no cumplir un plazo ocasionalmente. Ejemplo de estos sistemas son los dispositivos multimedia para el hogar y ciertos dispositivos embebidos como los controladores de hornos de microondas o de teléfonos IP [4].

1.2. Tiempo real estricto

Por otra parte, se llama sistema de tiempo real estricto a un sistema que debe cumplir forzosamente con los plazos establecidos para garantizar el correcto cumplimiento de los objetivos del sistema. Dentro de los sistemas de tiempo real estricto se identifican adicionalmente los sistemas de seguridad crítica [4], los cuales requieren de un diseño muy cuidadoso para garantizar el funcionamiento correcto y estable del sistema. Ejemplos de sistemas de tiempo real estricto son los ya mencionados controladores de robots industriales. Ejemplos de sistemas de misión crítica son los controladores de frenos en autos computarizados y los dispositivos embebidos relacionados a la salud como los marcapasos [4].

2. Características

Para garantizar el correcto funcionamiento de un sistema de tiempo real, un sistema operativo de tiempo real debe poseer tres características o propiedades importantes que deben guiar su diseño. La primera de estas, el determinismo, se relaciona con la importancia del tiempo en estos sistemas y es de crucial importancia en los sistemas de tiempo real estricto y los sistemas de seguridad crítica [4]. Las otras dos características son altamente deseables en cualquier sistema operativo, aunque cobran igualmente una enorme importancia en los sistemas de seguridad crítica.

2.1. Determinismo

Los sistemas de tiempo real real se diseñan para procesar y dar respuesta ante dos tipos de eventos externos, los eventos periódicos y aperiódicos. Se dice que un evento es periódico cuando este se produce de manera regular en intervalos de tiempo regulares. Por otra parte los eventos aperiódicos son de naturaleza impredecible y pueden ocurrir en cualquier momento [2]. Ambos tipos de eventos pueden ser producidos por el sistema mismo o por entes externos. La naturaleza de los eventos que debe procesar el sistema determina factores de diseño como el tipo de mecanismo de planificación de procesos e incluso la elección y entonado de los algoritmos de planificación en particular [2].

Para el caso de un sistema encargado de procesar eventos periódicos se puede aprovechar la naturaleza determinista de los eventos para simplificar el diseño del sistema utilizando mecanismos de planificación estática [2], los cuales establecen la planificación del sistema de forma fija durante la fase de diseño y desarrollo del sistema, o que calculan un orden de planificación durante el arranque del sistema. Este tipo de planificación se estudia más a fondo en la Sección 4.1.

En cambio, si el sistema debe procesar eventos aperiódicos entonces el sistema debe diseñarse de tal forma que el retraso introducido por las distintas tareas del sistema (como por ejemplo el despacho de procesos o el cambio de contexto) sea constante y por lo tanto predecible [4].

2.2. Estabilidad

La estabilidad o robustez de un sistema se define como la capacidad de un sistema de mantener su confiabilidad (véase la Sección 2.3) incluso frente a la ocurrencia de fallos críticos [7]. Se define como fallo crítico o maligno a aquel cuyo costo es órdenes de magnitud mayor a los beneficios que se pueden adquirir del uso regular del sistema. Un ejemplo de fallo crítico es el choque de un automóvil debido a la falta de respuesta de un sistema de frenos computarizado [7]. Existen múltiples estrategias que se pueden seguir para diseñar y construir un sistema de tiempo real estable, las cuales se detallan en la Sección 3.2.

2.3. Confiabilidad

Se define como confiabilidad a "la probabilidad de que un sistema provea un servicio específico hasta un tiempo t, dado que el sistema está operativo desde el comienzo, es decir, en un tiempo t_0" [7]. La confiabilidad se mide con una unidad llamada FIT (*Failure in Time* - Falla en el Tiempo), siendo 1 FIT igual a una falla en cada 115,000 años [7].

Para medir la confiabilidad C de un sistema en un tiempo t dado en horas se hace uso de la ecuación 1, donde λ representa la tasa de fallos por hora del sistema [7]. El inverso de la tasa de fallos por hora, es decir $\frac{1}{\lambda}$ se conoce como el MTTF (*Mean Time to Failure* - Tiempo Medio de Falla) del sistema.

$$C(t) = e^{-\lambda(t-t_0)} \tag{1}$$

Se llama sistema de confiabilidad ultra alta a un sistema que posee como requerimiento una tasa de fallos menor o igual a 10^{-9} fallos por hora, o lo que es lo mismo, una falla cada mil millones de horas [7].

3. Consideraciones de diseño

Al momento de diseñar un sistema operativo de tiempo real es necesario tomar en consideración ciertos factores que determinan el funcionamiento de componentes cruciales del sistema. El primero de estos se refiere al tipo de eventos a los que deben responder los procesos, los cuales determinan si estos son procesos periódicos o aperiódicos. El segundo factor a considerar consiste en las estrategias de tolerancia a fallos que debe poseer el sistema, en particular si se trata de un sistema de seguridad crítica.

3.1. Tipo de respuesta

En un sistema de tiempo real se pueden producir dos tipos de eventos que deben ser atendidos por los procesos del sistema. El primero de estos son los eventos de reloj del sistema. El segundo tipo de eventos corresponde a, valga la redundancia, eventos asíncronos producidos de forma externa al sistema [7].

3.1.1. Disparo de temporizadores

Un sistema operativo de tiempo real encargado de procesar eventos generados por un reloj interno se conoce como un sistema TT (*Time Triggered* - Activado por Tiempo) [7]. Los sistemas TT solo generan y responden a interrupciones regulares del reloj, las cuales se utilizan para iniciar cualquier actividad del sistema, como por ejemplo el planificar un proceso o realizar un intercambio de mensajes entre procesos [7]. Los sistemas TT se suelen componer principalmente de procesos periódicos [7].

3.1.2. Captura de eventos (sensores)

Por otra parte, un sistema operativo de tiempo real capaz de procesar interrupciones generadas por eventos externos e internos (lo que puede incluir la interrupción del reloj) se conoce como un sistema ET (*Event Triggered* - Activado por Eventos). Los sistemas ET suelen hacer uso de sensores que permiten al sistema el reaccionar a eventos físicos mediante un mecanismo de gestión de interrupciones clásico [7]. En estos sistemas se pueden presentar tanto procesos periódicos como aperiódicos.

3.2. Tolerancia a fallos

Como se mencionó en la Sección 2.2, un sistema robusto es aquel que puede mantenerse confiable incluso ante la presencia de fallos críticos. Para lograr esto se utilizan distintas herramientas de tolerancia a fallos con el objetivo de incrementar la probabilidad de que el sistema se mantenga estable ante estas posibles eventualidades. Estas estrategias se pueden aplicar tanto a nivel del diseño del sistema como a nivel de operación del sistema.

3.2.1. Diseño y verificación

Un sistema de tiempo real tolerante a fallos debe cumplir con los siguientes requisitos de diseño, los cuales están pensados para ser evaluados y certificados por una entidad externa e independiente a la encargada de diseñar y construir el sistema [7].

1. Los subsistemas que son críticos para la operación correcta del sistema debe estar protegidos por mecanismos de contención de fallas que eliminen la posibilidad de que los errores se propaguen desde otros subsistemas hacia estos subsistemas críticos.

2. Desde el punto de vista del diseño, todos los escenarios cubiertos por las hipótesis de carga y falla pueden ser manejados por la especificación sin necesidad de depender de argumentos probabilísticos.

3. La arquitectura del sistema soporta un proceso de verificación modular, en el cual cada subsistema puede ser verificado de forma independiente a los demás. A nivel de sistema solo debería ser necesario verificar los comportamientos emergentes de la interacción de los subsistemas.

Se entiende por hipótesis de carga y de falla a la especificación de los limites de carga pico y de fallas benignas y malignas que el sistema debe poder soportar en todo momento [7].

3.2.2. Estados seguros

A nivel operativo, un sistema de tiempo real puede hacer uso de un concepto conocido como estado seguro el cual se define como un estado conocido en el cual incluso la ocurrencia de una falla no puede provocar daños al sistema o a sus usuarios. Un sistema capaz de

identificar y alcanzar uno de estos estados seguros ante la ocurrencia de una falla crítica se conoce como un sistema *fail-safe* (seguro en falla) [7]. Un ejemplo de un sistema de esta categoría es el sistema de señalización de una red de trenes, en la cual es posible ordenar a todos los trenes a detenerse en un momento dado en caso de detectar una avería en la vía.

Para implementar este mecanismo se suele colocar un segundo sistema externo conocido como el *watchdog* (perro guardián), el cual se encarga de verificar el funcionamiento del sistema de tiempo real mediante la recepción de una señal conocida emitida por este último, llamada señal de vida. El *watchdog* esperará recibir esta señal dentro de un período de tiempo definido. En caso de no recibirse la señal, el *watchdog* deberá asumir que el sistema ha fallado y deberá forzar al sistema de tiempo real a pasar a un estado seguro [7].

En caso de que no sea posible identificar estados seguros para un sistema, este deberá ser diseñado de alguna manera para garantizar que el mismo pueda permanecer usable incluso a un nivel mínimo que permita a un operador el tratar de recuperar el correcto funcionamiento del sistema. Un sistema con estas características se dice que es *fail-operational* (operativo en falla). Un ejemplo de un sistema de este tipo es el piloto automático y sistema de control de vuelo de un avión comercial.

4. Planificación de procesos de tiempo real

En un sistema operativo de tiempo real existen dos enfoques de planificación de procesos que dependen del tipo de los procesos a planificar. Así mismo, existen otros dos enfoques los cuales dependen de las capacidades del sistema operativo con respecto a los procesos que se encuentran actualmente en ejecución.

4.1. Enfoque estático vs dinámico

Con respecto a los procesos del sistema se definen dos enfoques de planificación los cuales dependen de si dichos procesos son periódicos o aperiódicos, además de que dependen de la información que se posee sobre los mismos al momento del inicio del sistema.

El primero de estos enfoques se conoce como el enfoque estático y consiste en la toma de decisiones de planificación al momento del arranque del sistema operativo, decisiones que una vez son establecidas no pueden ser modificadas [2]. Este enfoque tiene la ventaja de ser altamente eficiente y se adapta principalmente a la planificación de procesos periódicos. Sin embargo, la planificación estática sufre de un gran problema, puesto que cualquier algoritmo de planificación basado en este enfoque necesita poseer de antemano información completa sobre cada proceso que puede existir en el sistema y sobre los plazos que deben cumplir estos.

Para que un sistema de tiempo real basado en procesos periódicos sea capaz de utilizar un algoritmo de planificación estático, este debe ser un sistema planificable [2]. Se dice que un sistema de tiempo real basado en procesos periódicos es planificable si cumple la relación establecida en la Inecuación 2. Esta inecuación indica que si existen m eventos periódicos en el sistema tales que cada evento i tiene un período P_i y necesita de T_i unidades de tiempo de CPU (*Central Processing Unit* - Unidad Central de Procesos) para ser tratado, entonces el

tiempo necesario para atender cada evento debe ser menor o igual al período de ocurrencia de dicho evento, asumiendo que el retraso de planificación y despacho de procesos del sistema operativo es despreciable [2]. En otras palabras, la Inecuación 2 establece que la tasa de uso del CPU por parte de los procesos periódicos no debe exceder el 100 % [2]. Un ejemplo de un algoritmo de planificación estático se detalla en la Sección 4.3.

$$\sum_{i=1}^{m} \frac{T_i}{P_i} \leq 1 \tag{2}$$

Por otra parte el segundo enfoque de planificación de procesos se conoce como el enfoque dinámico. En este enfoque las decisiones de planificación son tomadas por el sistema operativo en tiempo de ejecución [2]. Un ejemplo de un algoritmo de planificación dinámico se describe en la Sección 4.4.

4.2. Enfoque apropiativo vs no apropiativo

Con respecto al sistema operativo existen dos enfoques de planificación de procesos. El primero se conoce como el enfoque apropiativo y consiste en la propiedad del sistema operativo de desalojar del CPU un proceso que se encuentre en ejecución a favor de otro proceso distinto según algún criterio que permita tomar esa decisión [2]. El enfoque contrario consisten en no otorgar esta habilidad al sistema operativo y se conoce como el enfoque de planificación no apropiativo [2].

4.2.1. Planificación basada en prioridades e inversión de prioridades

La clase de planificación de procesos más utilizada en los sistemas operativos de tiempo real es la de los algoritmos de planificación por prioridades [2]. Estos algoritmos se caracterizan por asociar un valor numérico llamado prioridad a cada proceso del sistema, el cual se utiliza para determinar cual proceso ejecutar a continuación. El sistema operativo siempre ejecutará el proceso con la prioridad mayor [2].

Los algoritmos basados en prioridades son algoritmos apropiativos [2] y como tales son sensibles a un problema conocido como inversión de prioridades [8]. Este problema se presenta de forma general en cualquier sistema operativo que posea mecanismos de exclusión mutua para acceso a recursos y protección de datos compartidos tales como semáforos o monitores además de utilizar planificación de procesos basada en prioridades [2].

El problema de la inversión de prioridades se presenta cuando un proceso con una cierta prioridad impide que un proceso de prioridad mayor se ejecute cuando este lo necesite debido a que ambos procesos están intentando obtener acceso a una misma sección crítica, la cual está siendo protegida mediante algún mecanismo de exclusión mutua [8]. Considérese el siguiente ejemplo: existen tres procesos dentro del sistema llamados A, B y C tales que las prioridades respectivas P_i a estos procesos cumplen que $P_A > P_B > P_C$. Supongamos que el proceso C obtiene acceso a una sección crítica durante su ejecución. Supongamos igualmente que el proceso A intentará obtener acceso a la misma sección crítica, acceso que no podrá

obtener dado que C se encuentra actualmente en dicha sección. Debido a esto el proceso A se bloqueará hasta que C libere la sección crítica. Si en este momento comienza su ejecución el proceso B, este tomará control del sistema dado que posee una prioridad mayor que C. En este momento el proceso B ha usurpado efectivamente la prioridad de A, impidiendo que este último proceso se ejecute de forma indefinida [2].

Una situación de inversión de prioridades causó que en la década de 1990 el robot de exploración Mars Rover *de la NASA* (National Aeronautics and Space Administration - *Administración Nacional de la Aeronaútica y el Espacio*) *fallara en su misión hasta tanto no fue reparado de forma remota una vez este había sido desplegado en el planeta Marte [8].*

Una solución al problema de la inversión de prioridades es el uso de un mecanismo de herencia de prioridades [2]. Con este mecanismo la herramienta de exclusión mutua utilizada por los procesos tiene que poseer una prioridad la cual inicialmente se coloca en la prioridad mínima del sistema. Cada vez que un proceso solicite acceso a la sección crítica protegida por el mecanismo de exclusión mutua en cuestión, este tomará la prioridad de dicho proceso si esta es mayor a la prioridad que tiene el mecanismo asignada actualmente. De esta forma el proceso que obtenga acceso a la sección crítica modificará su prioridad según la que esté establecida en el mecanismo de exclusión mutua hasta el momento en que libere la sección crítica.

Si aplicamos el mecanismo de herencia de prioridades al ejemplo anterior veremos que cuando el proceso A solicite acceso a la sección crítica establecerá su prioridad al mecanismo de exclusión mutua, prioridad que será heredada por el proceso C. De esta manera, cuando el proceso B pase a estado de listo no podrá ejecutarse dado que el proceso C ahora posee una prioridad mayor. Eventualmente el proceso C deberá liberar la sección crítica permitiendo que el proceso A pueda ejecutarse [2].

4.3. Planificación de tasa monotónica

El algoritmo RMS (*Rate Monotonic Scheduling* - Planificación de Tasa Monotónica) fue propuesto en 1973 por Liu y Layland en [9]. Es un algoritmo estático y apropiativo que se utiliza para modelar la planificación de procesos periódicos en sistemas que cumplan las siguientes condiciones [2]:

1. Todo proceso periódico debe completar su trabajo dentro de su período de ejecución.
2. Todos los procesos son independientes.
3. Cada proceso utilizará el mismo tiempo de CPU en cada ráfaga.
4. Ningún proceso aperiódico debe cumplir plazos de tiempo.
5. La sobrecarga de la apropiación de procesos es despreciable.

A cada proceso periódico se le asigna una prioridad igual a la frecuencia con la cual se activa este proceso [2]. Por ejemplo, un proceso que tiene un período de 30 milisegundos

tiene una frecuencia de 33 activaciones por segundo, por lo tanto su prioridad será 33. Con este esquema las prioridades de los procesos se establecen de forma lineal con su respectiva tasa de ejecución, lo que le da su nombre al algoritmo. Estas prioridades se establecen al iniciar el sistema [2]. Durante la ejecución se planifica siempre al proceso de mayor prioridad (un número mayor estable una prioridad más alta), sacando del CPU al proceso que se esté ejecutando actualmente de ser necesario [2]. Liu y Layland demostraron que este algoritmo es óptimo entre los algoritmos de planificación estáticos.

Para garantizar que este algoritmo funcione correctamente, la tasa de utilización del sistema debe cumplir la relación establecida por la Inecuación 3, donde m representa la cantidad de procesos periódicos en el sistema y para cada proceso periódico i, T_i representa la duración de una ráfaga de CPU y P_i el período de dicho proceso. Este límite fue igualmente demostrado por Liu y Layland en [9]. Si la tasa de utilización del CPU establece este límite entonces el algoritmo puede fallar en planificar ciertos conjuntos de procesos aunque puede funcionar para otros conjuntos diferentes [2].

$$\sum_{i=1}^{m} \frac{T_i}{P_i} \leq m(2^{1/m} - 1) \tag{3}$$

La Figura 1 muestra una corrida de ejemplo del algoritmo RMS en un sistema con tres procesos denominados A, B y C. Los tres procesos son periódicos y tienen los períodos y duración de ráfaga de CPU visibles en la Figura en los diagramas de Gantt identificados con el nombre de cada proceso.

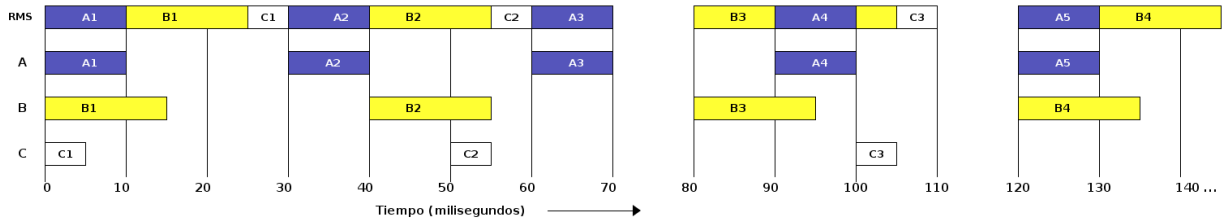

Figura 1: Ejemplo de planificación de procesos con RMS. Figura recuperada de [2].

4.4. Planificación de Plazo más Temprano Primero

EDF es un algoritmo de planificación dinámico y apropiativo capaz de planificar tanto procesos periódicos como aperiódicos, el cual se utiliza como una alternativa a RMS [2].

Con este algoritmo el sistema operativo mantiene una cola con todos los procesos activos, los cuales están ordenados por el tiempo en el cual se vencen sus plazos respectivos, funcionando este tiempo como la prioridad de cada proceso. Para poder ordenar la cola es necesario que cada proceso indique el tiempo faltante para el vencimiento de su plazo al momento en que este pase a estado de listo. El sistema operativo siempre deberá ejecutar el proceso que se encuentre al frente de la cola [2]. Si un proceso pasa a estado de listo mientras

otro proceso está en ejecución, y el plazo del proceso recién llegado se vence antes del plazo del proceso activo, entonces el sistema operativo debe realizar un cambio de contexto para colocar al nuevo proceso en ejecución. La Figura 2 muestra un ejemplo de planificación con el algoritmo EDF en comparación con el algoritmo RMS descrito en la Sección anterior.

La Figura 2 consiste en varios diagramas de Gantt los cuales ilustran una comparación entre los algoritmos RMS y EDF. En este ejemplo existen 3 procesos periódicos identificados como A, B y C los cuales tienen los períodos y ráfagas de CPU visibles en los correspondientes diagramas de Gantt de la Figura. Se asume que los tres procesos se encuentran disponibles para ser ejecutados a partir del tiempo 0. Como se puede apreciar, el algoritmo RMS falla para este ejemplo puesto que a los 50 milisegundos el proceso C ha vencido su primer plazo pero nunca fue ejecutado. Como se puede constatar utilizando la Inecuación 3, el límite de la tasa de utilización del CPU para estos 3 procesos es de aproximadamente 0,78, mientras que la tasa de utilización real es de aproximadamente 0,975.

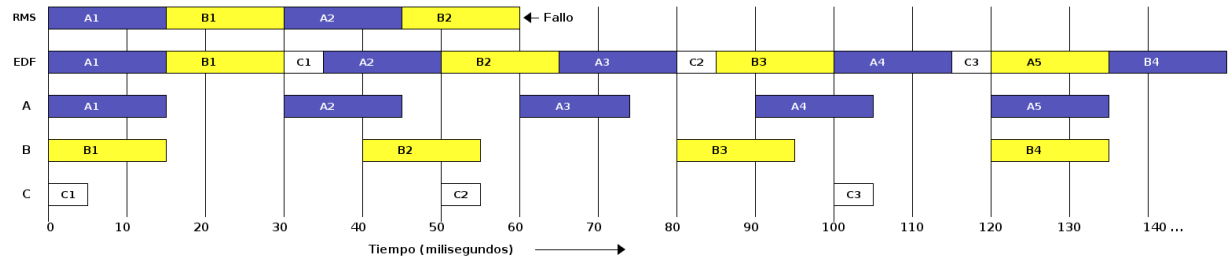

Figura 2: Ejemplo de planificación de procesos con RMS y EDF. Figura recuperada de [2].

Conclusiones

En este documento se realizó un estudio sobre los sistemas operativos de tiempo real, los cuales se definen como sistemas computacionales encargados de proveer respuestas o realizar determinados procesos dentro de plazos de tiempo específicos que pueden ser muy estrictos. Estos sistemas se organizan en dos grandes categorías. La primera es la de los sistemas de tiempo real suave en los cuales es posible no cumplir plazos ocasionalmente sin provocar daños o pérdidas. En contraposición, los sistemas de tiempo real estricto son aquellos sistemas caracterizados por implicar fallas catastróficas ante el incumplimiento de los plazos establecidos. Estos últimos a su vez se pueden subdividir en los sistemas de seguridad crítica que tiene requisitos sumamente altos de estabilidad y confiabilidad.

Un sistema de tiempo real puede atender eventos periódicos generados por un reloj interno, o eventos producidos por entes externos al sistema los cuales son capturados por sensores y reportados al sistema mediante el mecanismo tradicional de interrupciones. Para atender estos eventos se definen entonces dos tipos de procesos, los periódicos y los aperiódicos respectivamente. Dadas las diferentes naturalezas de estos procesos se hace necesario definir enfoques que permitan planificarlos y ejecutarlos correctamente. El primero de estos

enfoques se conoce como planificación estática, la cual se utiliza para planificar procesos periódicos. El segundo enfoque se llama planificación dinámica y se utiliza para planificar ambos tipos de procesos.

La planificación en los sistemas de tiempo real es principalmente apropiativa y basada en prioridades. En otras palabras, a cada proceso el sistema operativo le asigna un valor numérico que indica su prioridad, la cual se utiliza para desalojar procesos en ejecución cuando se activa un proceso de mayor prioridad. Ejemplos de este tipo de mecanismo de planificación son los algoritmos RMS, un algoritmo estático, y el algoritmo EDF, un algoritmo dinámico.

Referencias

[1] D. Perez Abreu, *Sistemas Embebidos y Sistemas Operativos Embebidos*, Notas de Docencia ND 2009-03, Esc. de Computación, Fac. de Ciencias, Universidad Central de Venezuela, 2009.

[2] A. Tanenbaum, *Modern Operating Systems*, 3^a Edición, Pearson, 2008.

[3] A. Tanenbaum y A. Woodhull, *Operating Systems: Design and Implementation*, 3^a Edición, Pearson, 2006.

[4] A. Silberschatz, P. Galvin y G. Gagne, *Fundamentos de Sistemas Operativos*, 7^a Edición, McGraw Hill, 2006.

[5] G. Galeano, *Programación de Sistemas Embebidos en C*, 1^a Edición, Alfaomega, 2009.

[6] A. Barbalace et al., *Performance comparison of VxWorks, Linux, RTAI and Xenomai in a hard real-time application*, 15th IEEE-NPSS Real-Time Conference, pp. 1-5, 2007.

[7] , H. Kopetz, *Real-time systems: design principles for distributed embedded applications*, Springer Science & Business Media, 2011.

[8] M. Jones, *What Really Happened on Mars Rover Pathfinder*, The Risks Digest, Vol. 19, No. 49, pp. 1-2, 1997.

[9] C. Liu y J. Layland, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, Journal of the ACM, Vol. 20, No. 1, pp. 46-61, 1973.