Kernel de Tiempo Real para Control de Procesos

Oscar Miranda Gómez, Pedro Mejía Álvarez
CINVESTAV-IPN, Sección de Computación
Av. IPN No. 2508, México, D. F. 07300.
omiranda@computacion.cs.cinvestav.mx, pmejia@delta.cs.cinvestav.mx

Resumen

En este artículo se describe la estructura de un Kernel de Tiempo Real desarrollado en la Sección de Computación del CINVESTAV-IPN. Las ventajas de este sistema operativo de tiempo real son principalmente su tamaño, su modularidad, su facilidad de poder comunicarse con otros Kernels de tiempo real ejecutándose en otras computadoras y su interfaz con un sistema gráfico de simulación de tareas de tiempo real.

El Kernel de Tiempo Real es pequeño (de tamaño 20.5Kb) y además realiza funciones que todo sistema en tiempo real requiere, como son: a) manejo de interrupciones externas, b) ejecución de tareas concurrentes, c) comunicación y sincronización entre procesos mediante las primitivas de acceso a buzones y semáforos, d) planificación de tareas periódicas y aperiódicas, mediante las políticas de planificación Rate Monotonic y Earliest Deadline First (EDF) [6], y e) manejo de tiempos.

Palabras Clave

Sistemas en Tiempo Real, Kernel, Rate Monotonic, EDF, Planificación de tareas.

1. Introducción

En la actualidad el avance de la tecnología ha permitido una miniaturización en los dispositivos electrónicos y un incremento en la capacidad de procesamiento de estos dispositivos. Estos dispositivos se encuentran en general empotrados (embebidos) dentro de aplicaciones tales como teléfonos celulares, PALM's, computadoras industriales, robots, satélites, automóviles y también en distintos aparatos electro-domésticos. La mayoría de estos dispositivos contienen un procesador y un pequeño sistema operativo empotrado el cual es capaz de controlar todo el hardware de manera eficiente. Debido a la naturaleza de estos dispositivos embebidos, en ocasiones se les demanda no solo un correcto y eficiente funcionamiento sino también un estricto cumplimiento de requerimientos temporales. A estos sistemas se les conoce como sistemas de tiempo real embebidos.

En un sistema de tiempo real el principal componente lo constituye el sistema operativo o Kernel. Un sistema operativo de tiempo real se ejecuta por lo general sobre un plataforma embebida (que puede ser un microcontrolador, un DSP o cualquier procesador convencional). Este sistema operativo debe ser capaz de controlar todos los recursos de hardware de la plataforma que se encuentra y también de administrar todos los tiempos de ejecución de las tareas de tiempo real. Normalmente este sistema operativo no maneja discos, memoria cache, DMA o complejos sistemas de redes de comunicaciones, debido a que su ejecución debe ser predecible (debe ser posible estimar con exactitud sus tiempos de ejecución).

Las tareas de tiempo real son procedimientos secuenciales que se ejecutan en forma *concurrente* con otras tareas del sistema. A una tarea de tiempo real τ_i se le asocia, un tiempo de arribo a_i, un período de ejecución, T_i y un plazo de respuesta D_i [1]. Estas tareas se clasifican en *periódicas* y *aperiódicas*. Las tareas periódicas se ejecutan a intervalos regulares (llamados períodos), mientras que las tareas aperiódicas no tienen un tiempo de arribo conocido, se ejecutan por un solo período de ejecución, y cuando arriban deben de cumplir con sus plazos de respuesta. Una tarea de tiempo real que controla procesos críticos (en los cuales no se permite la pérdida de su plazo de respuesta) se le conoce como tarea crítica. A las tareas acríticas se les exige terminar lo más pronto posible su ejecución, pero como no es crítico pueden llegar a perder algunos de sus plazos de respuesta.

Actualmente el Kernel se ejecuta sobre MS-DOS, sin embargo, ha sido diseñado como software independiente (portable) del sistema operativo, con el fin de que pueda cargarse en cualquier plataforma que contenga procesadores INTEL o compatibles. Se tiene contemplado que el Kernel sea totalmente independiente, siendo este el que se encargue de controlar todo el hardware y procesos que se encuentren en la máquina, o en su caso, en el sistema empotrado en donde se implemente.

2. Trabajo Relacionado

Existen en la actualidad un gran número de sistemas operativos de tiempo real industriales y experimentales [8, 3].

Entre los sistemas operativos comerciales más uti-

lizados están LinxOs, VxWorks, QNX, RT-Linux, y TimeSys Linux. Estos sistemas operativos incluyen una amplia gama de servicios tales como manejo de procesos, manejo de memoria, manejo de tiempos, manejo de dispositivos e interfaces con distintas plataformas de hardware [10]. Por lo general estos sistemas operativos son de gran tamaño, y están diseñados para cubrir una amplia gama de aplicaciones de sistemas embebidos.

Por otro lado, existen los sistemas operativos experimentales de los cuales los más conocidos en la actualidad son: EMERALDS de la Universidad de Michigan [11], S.H.A.R.K de la Escuela Superior Santa Ana de Pisa Italia, MACH de la Universidad Carnegie Mellon, MARRUTTI de la Universidad de Maryland, SPRING de la Universidad de Massachusetts en Amherts y MARS de la Universidad Técnica de Viena. Todos estos sistemas operativos incluyen distintos métodos de planificación, son de tamaño medio a pequeño, incluyen interfaces gráficas y son bastante flexibles en su diseño. En varios de estos sistemas operativos se incluyen técnicas desarrolladas por dichas Universidades, tales son los casos de MARS con su técnica de planificación basada en eventos de tiempos, o SPRING que se basa en la *técnica de planificación del mejor esfuerzo* (best effort scheduling).

3. Estructura del Kernel

El Kernel desarrollado es de tamaño pequeño, de 20.5KBytes, pero este tamaño puede variar dependiendo de los servicios, librerías y controladores que se le pudieran añadir.

El Kernel esta compuesto de los siguientes componentes:

- Manejador de Procesos.
- Manejador de Buzones para comunicación síncrona y asíncrona.
- Manejador de Tareas Periódicas y Aperiódicas.
- Manejador de Tiempos.
- Manejador de Interrupciones externas.
- Manejador de Semáforos para sincronización entre procesos.
- Manejador de Comunicaciones externas.
- Mecanismos de Planificación Rate Monotonic, EDF y FIFO Round Robin.
- Interface para Depuración y Monitorización.

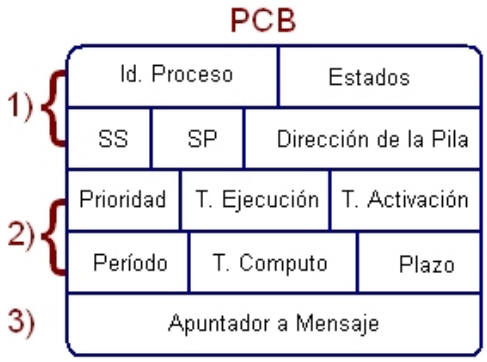

Figura 1: Bloque de control de procesos (Process Block Control)

3.1. Manejador de Procesos

El manejador de procesos (tareas) es uno de los servicios básicos con los que cuenta el Kernel y ofrece varias funciones de soporte tales como la creación y eliminación de procesos, la planificación de tareas (EDF, Rate Monotonic, FIFO Round Robin) y el cambio de contexto, [2].

Un *proceso* es un conjunto de código secuencial identificado con un nombre y una prioridad, el cual se ejecuta concurrentemente con otros procesos [4]. Cada proceso tiene asociada una estructura llamada PCB (Process Control Block) la cual contiene la información necesaria para el control del proceso.

La estructura del PCB en el Kernel está representada en la Figura 1, en la cual se puede apreciar que la información está ordenada de la siguiente manera:

1) Información del proceso.- Aquí se encuentra almacenado el identificador del proceso, el *estado* en que se encuentra durante la ejecución del Kernel (listo, suspendido, etc.) y la dirección de la pila que le corresponde al proceso. La pila almacena el contenido de sus registros (cs, ip, bp, di, si, ds, es, dx, cx, bx, ax y flags) así como las direcciones de las instrucciones que ejecuta el proceso. Los registros incluidos en el stack son indispensables para que el Kernel pueda llevar a cabo el *cambio de contexto*.

2) Información para el planificador.- Aquí se encuentran los datos necesarios para que el proceso pueda ser planificado. Contiene información como la prioridad, el tiempo de cómputo, el plazo, el tiempo de activación y el tiempo de ejecución. Estos datos son necesarios para que las políticas de planificación puedan calcular cuando le corresponde ejecutarse a cada proceso.

3) Información de memoria utilizada en buzones.- Sólo contiene un apuntador a una dirección de memoria reservada para almacenar un mensaje

del buzón en caso de que el proceso sea introducido a la cola de procesos bloqueados por buzón (cpbb).

El Kernel inicialmente esta configurado para trabajar con un máximo de 32 procesos, sin embargo el usuario en base a sus necesidades, puede aumentar el número de los procesos a controlar por el Kernel. Los procesos reciben una prioridad de ejecución que depende de la política de planificación en la que se ejecuten. La prioridad de una tarea es estática si esta se ejecuta bajo la política Rate Monotonic o Fifo Round Robin, y es dinámica cuando se ejecuta con la política EDF. Existen dos procesos importantes en el Kernel llamados: *primero y ultimo* los cuales son activados al iniciar el Kernel. El proceso primero es el encargado de activar a los demás procesos que se van a ejecutar en el sistema y el proceso ultimo es un proceso que se ejecuta cuando ningún otro proceso se encuentra listo para ser ejecutado.

3.1.1. Estados de los Procesos

Cada proceso puede encontrarse en cualquiera de los siguientes estados [7], como se describe en la Figura 2:

Ejecución.- En este estado, el proceso tiene el control del procesador.

Listo.- En este estado, el proceso se encuentra listo para ejecutarse y espera a que se le asigne el procesador.

Bloqueado.- En este estado, el proceso se encuentra bloqueado en algún recurso, como puede ser un semáforo, o un un buzón.

Retrasado.- En este estado el proceso se encuentra bloqueado, esperando por un tiempo determinado antes de regresar al estado de listo.

Eliminado.- En este estado se encuentran los procesos que no han sido activados o que ya han terminado su ejecución.

Los procesos cambian de estado por alguna de las siguientes razones (ver Figura 2):

1. el proceso es creado.
2. se le asigna el procesador.
3. se le quita el procesador debido a: un bloqueo, el alistamiento de un proceso con mayor prioridad o por el fin de su tiempo de cómputo.
4. el proceso se retrasa.
5. ocurre una vez que termina su tiempo de retraso, quedando listo para ejecutarse.

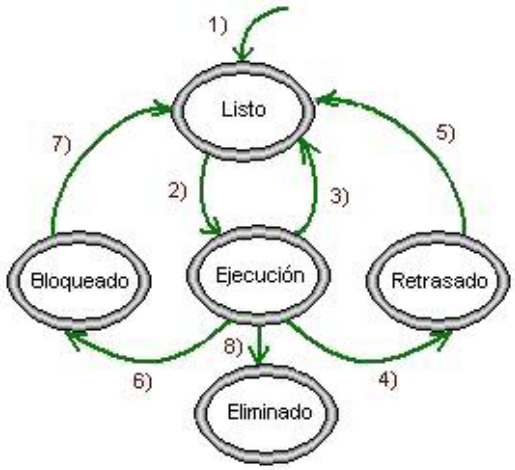

Figura 2: Estados de los Procesos

6. el proceso se bloquea por un recurso o interrupción.
7. obtuvo el recurso que esperaba y queda listo para ejecutarse.
8. el proceso se elimina a sí mismo.

3.1.2. Manejo de Colas de Procesos

Debido a que el Kernel se ejecuta sobre un sólo procesador (aunque tiene también la capacidad de comunicarse con otros procesadores sobre los que se ejecute el Kernel), solo un proceso puede estar ejecutándose en el procesador, mientras que varios procesos pueden encontrarse listos para ejecución y otros más pueden estar bloqueados por recursos.

Dependiendo del estado en que se encuentren las tareas, cada una de estas puede encontrarse en alguna de las siguientes colas:

1. Cola de procesos listos.
2. Cola de procesos bloqueados por enviar o recibir un mensaje.
3. Cola de procesos bloqueados por espera en un semáforo.
4. Cola de procesos retrasados.

Las colas de procesos fueron diseñadas con el fín de manejar eficientemente la ejecución de las tareas de acuerdo a su estado. Las tres primeras colas tienen una estructura similar con múltiples niveles de prioridades, como se muestra en la Figura 3. La primera tarea en la cola es aquella que cuenta con la mayor prioridad en el sistema. En nuestro Kernel, asumimos que los procesos se insertan al final de la cola respectiva dependiendo de su prioridad y su arribo a la cola.

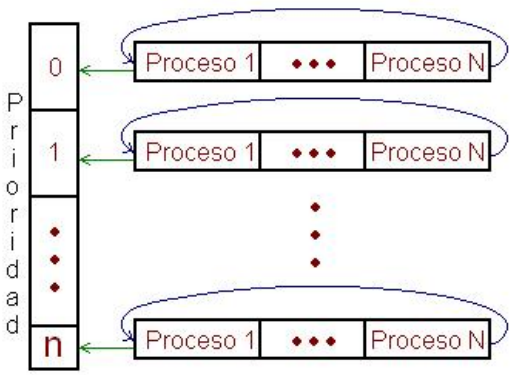

Figura 3: Colas de Procesos

3.2. Manejador de Interrupciones

Actualmente el Kernel procesa solo las solicitudes de interrupción del reloj y del teclado. La interrupción 8 es capturada con el fín de manejar el reloj del Kernel. Esta interrupción fue re-programada a 1 milisegundo con la finalidad de aumentar la granularidad del sistema. De esta forma, el *Quantum*, o el tiempo máximo asignado a una tarea para su ejecución, cuenta con mayor precisión. Con esta interrupción el Kernel controla el cambio de contexto y el tiempo de bloqueo de la primitiva retrasa. La interrupción del teclado permite al Kernel interrumpir su ejecución, y es útil en la detección de caracteres del teclado.

3.3. Manejo del Tiempo

Mediante el manejador del tiempo, el Kernel puede manejar el tiempo en que una tarea se ejecuta en forma continua. A este tiempo se le conoce como *Quantum* y depende de la resolución del timer del sistema. Como se explicó anteriormente, se modificó el timer del sistema para lograr una resolución aproximada de 1 milisegundo. Por esta razón, el Quantum solo puede darse en unidades de 1 milisegundo.

Dentro del manejo del tiempo también se permite que una tarea pueda dormirse durante un cierto tiempo mediante la primitiva *retrasa*. Con esta primitiva un proceso es incluido a una cola de procesos dormidos, y despertará hasta que se consuma el tiempo que solicita. El parámetro que recibe la primitiva retrasa, permite al proceso que la ejecuta dormirse por un número dado de unidades de tiempo.

3.4. Sincronización y Comunicación entre Tareas

El Kernel contiene mecanismos de sincronización y comunicación entre tareas, los cuales se manejan mediante semáforos y buzones.

3.4.1. Semáforos

Mediante los *semáforos* es posible sincronizar y establecer regiones de exclusión mutua entre las tareas del sistema. Un semáforo es una variable entera que se crea en el espacio de memoria del Kernel [9], y cuyo valor sólo se puede manipular con tres primitivas que hacen sus operaciones de forma atómica[1].

Las tres primitivas de los semáforos son:

IniciaSemáforo.- En esta primitiva se le da un valor inicial (positivo) a la variable del semáforo, con lo cual se crean una cola asociada al semáforo. En esta cola, se incluirán los procesos bloqueados por este recurso.

Señal.- Esta primitiva permite incrementar en una unidad, a la variable del semáforo. Si la variable del semáforo es negativa indicará que existe algún proceso bloqueado (en la cola de este semáforo). Por lo tanto si al ejecutar la primitiva señal, el contador es negativo, algún proceso bloqueado en la cola de este semáforo (el que se encuentre al principio de la cola), se pasará a la cola de procesos listos. En este caso, el Kernel provocará un cambio de contexto para verificar si el proceso desbloqueado, es de mayor prioridad que el proceso en ejecución. Si al ejecutar la primitiva señal no existen procesos bloqueados (el contador es cero o positivo), se incrementa el contador del semáforo y el proceso continuará su ejecución.

Espera.- Esta primitiva permite decrementar en una unidad a la variable del semáforo. Si después de decrementar esta variable, su valor es negativo, el proceso que ejecuta esta primitiva es enviado a la cola del semáforo y sacado de ejecución. En este caso, el Kernel invocará a la rutina de cambio de contexto para ejecutar al siguiente proceso de la cola de listos. Por el contrario, si después de ejecutar la primitiva espera, la variable del semáforo contiene un valor cero o positivo, el proceso continuará su ejecución.

En la Figura 4 se muestran 2 tareas haciendo acceso de un semáforo para implementar regiones críticas. El uso de estas regiones críticas permite a cada proceso accesar una variable compartida en forma exclusiva.

3.4.2. Buzones

Un *buzón* es una área de memoria compartida capaz de contener un número limitado de mensajes. Estos

[1]Todas las primitivas del Kernel son de ejecución atómica, lo cual implica que no podrán ser interrumpidas durante su ejecución. Las interrupciones son deshabilitadas durante la ejecución de las primitivas a fin de proteger las variables y la memoria del Kernel

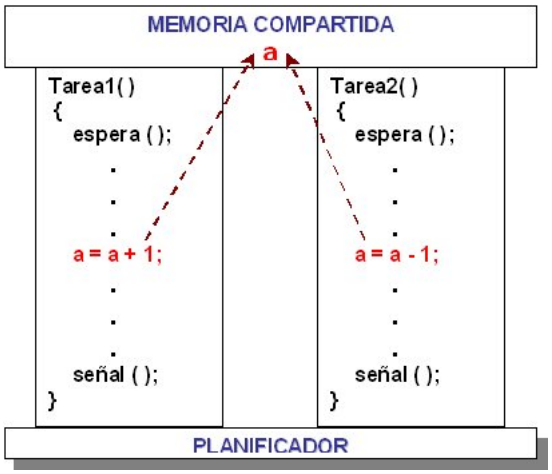

Figura 4: Utilización de los semáforos en el Kernel

mensajes son almacenados en el buzón mediante una cola circular tipo FIFO. Con los buzones es posible comunicar a dos o mas procesos entre si. Cada buzón tiene asociada una cola de procesos. En esta cola de procesos se introducirán aquellos procesos que se encuentren bloqueados por este buzón [9].

En el manejo de los buzones se utilizan tres operaciones básicas:

CrearBuzón.- Esta primitiva permite crear la estructura de datos que define al buzón y su cola de procesos asociada.

EnviarMensaje.- Mediante esta primitiva es posible enviar un mensaje en formato ASCII al buzón indicado (el cual previamente tuvo que haber sido creado). Si al enviar un mensaje alguna tarea se encontraba bloqueada esperando mensajes, esta tarea será desbloqueada y enviada a la cola de procesos listos.

Si un proceso envía un mensaje al buzón y el buzón se encuentra lleno, provocará que el proceso se incluya en la cola del buzón. Este proceso permanecerá en esta cola hasta que otro proceso reciba mensajes y libere suficiente espacio del buzón.

RecibirMensaje.- Esta primitiva permite recibir mensajes del buzón indicado. Si el buzón se encuentra vacío, el proceso que invoca a esta primitiva será bloqueado en la cola respectiva.

Note que la cola del buzón contendrá solo a procesos bloqueados esperando recibir mensajes, o solo a procesos esperando enviar mensajes al buzón. Por su implementación, los dos tipos de procesos no pueden coexistir en una misma cola.

3.5. Características Adicionales en el Kernel

3.5.1. Tamaño

Debido a que el Kernel fue diseñado para su implementacion en un sistema empotrado, el Kernel contiene sólo las funciones que son necesarias para el control de los sistemas de tiempo real en este tipo de dispositivos. Por lo tanto, el tamaño del Kernel es muy pequeño (20.5Kb). Esta característica hace posible que se pueda implementar en sistemas empotrados con altas restricciones de memoria. En la tabla 1 se muestra a detalle el tamaño de cada uno de los archivos que conforman el Kernel.

3.5.2. Granularidad

Como se explicó anteriormente, la granularidad del tiempo en el Kernel es de 1 milisegundo. Esto permite programar el Quantum de los procesos, y el tiempo de la primitiva retrasa, con esta resolución. Actualmente, el Kernel maneja la interrupción 8, para efectuar su cambio de contexto. Sin, embargo, el Kernel también es capaz de manejar otras interrupciones tales como la interrupción 70h, que pertenece al reloj de tiempo real de la PC.

De la misma forma, el control del tiempo e interrupciones periódicas dentro del Kernel se llevan a cabo mediante el temporizador 8254. Este temporizador, en cualquier PC con procesador compatible con Intel, inicialmente se encuentra alimentado con una señal de 1,193,180Hz y configurado en modo 2 con lo cual genera 18.2 ticks de reloj por segundo (una interrupción 08 es generada por cada tick). Esta frecuencia es muy baja para los sistemas de tiempo real, por lo que en el Kernel se re-programó el timer cambiando su valor inicial por otro que pudiera generar interrupciones periódicas a 1000Hz (1,000 por segundo); esta frecuencia es similar a la que se obtiene con el reloj de tiempo real (RTC) pero con la ventaja de que el timer tiene la mayor prioridad y ninguna otra interrupción generada en la máquina lo puede detener.

Nombre del archivo	*Tamaño*
Kernel	1.05Kb
config	231 bytes
constant	1.55Kb
errores	1.85Kb
libreria	1.97Kb
mancolas	4.24Kb
mancpu	491 bytes
manreg	1.49Kb
planif	1.99Kb
pribuz	2.51Kb
pripro	621 bytes
prisem	1.67Kb
pritim	255 bytes
teclado	304 bytes
timer	376 bytes
TOTAL	20.5Kb

Tabla 1: Tamaño del Kernel.

Primitiva	*Tiempo mínimo(μs)*	*Tiempo máximo(μs)*	*Tiempo promedio(μs)*
Inicialización del sistema	11.43	13.23	12.34
Cambio de contexto	30.37	34.12	32.44
Activa	11.73	12.57	11.93
Elimina	11.73	14.24	12.08
Retrasa	11.23	13.23	12.23
Crea semáforo	11.73	12.57	11.81
Señal	11.73	13.40	12.17
Espera	11.73	12.57	12.17
Crea buzón	11.73	12.57	11.83
Envía Mensaje	12.57	14.24	12.80
Recibe Mensaje	13.40	15.92	14.23
Inserta a la cola	11.73	12.57	12.32
Saca de la cola	11.73	12.57	12.30
Busca el mayor	23.46	24.30	23.59
Earliest Deadline First	11.73	15.08	12.28
Rate Monotonic	11.73	13.40	12.25

Tabla 2: Tiempos de ejecución de las primitivas del Kernel.

3.5.3. Tiempos de ejecución del Kernel

En cualquier aplicación de tiempo real es importante conocer el tiempo de ejecución de las primitivas diseñadas en el Kernel, debido a la predecibilidad con que debe contar el sistema. En la tabla 2 se indica que el cambio de contexto tarda en promedio 32.44 microsegundos. Debido a que la granularidad del Kernel es de 1 milisegundo (1,000 interrupciones por segundo), el tiempo que le resta al proceso para ejecutarse (después de la ejecución de la rutina de cambio de contexto) sería de aproximadamente 967.55 microsegundos.

Los tiempos presentados en la tabla 2 fueron obtenidos en una computadora Laptop Pentium 4 a 1.4GHz, con 240MB de RAM y 16MB de RAM en video. Con la finalidad de obtener las mediciones más realistas posibles, el Kernel se ejecutó solo bajo el ambiente de MS-DOS (fuera del ambiente de Windows).

4. Arquitectura del Kernel

En la Figura 5 se presenta la arquitectura general del Kernel en tiempo real. En el nivel más bajo se encuentran los distintos dispositivos que controla el Kernel. En el siguiente nivel están las distintas funciones del Kernel, conocidas como *primitivas*. En la parte superior, se encuentran los procesos del usuario quienes interaccionan con el Kernel. Entre el Kernel y los procesos del usuario se encuentra una interface la cual permite desligar a ambos componentes. Esta interface permite, mediante interrupciones, conectar a ambos componentes aunque estos hallan sido compilados por separado. Esta interface permite que los procesos de aplicación puedan estar en un lenguaje distinto al del Kernel.

El Kernel soporta una arquitectura de un procesador, en el cual los procesos se ejecutan concurrentemente. El Kernel se implementó en lenguaje de programación C y en Ensamblador, propios de la familia de procesadores Intel 80x86. Algunas partes del Kernel, consideradas como críticas, como son el cambio de contexto, el manejo de interrupciones y el copiado de mensajes, se codificaron en lenguaje ensamblador para optimizar sus tiempos de ejecución.

5. Planificación de Procesos

En las aplicaciones de Tiempo Real, lo importante es proporcionar un orden de ejecución a las tareas de tiempo real, de forma tal que todas estas cumplan con sus plazos de respuesta. Para esto, es necesario conocer con la mayor precisión posible los tiempos de arribo, los tiempos de ejecución y los tiempos de respuesta, de cada una de las tareas del sistema. Así mismo es necesario conocer con precisión los tiempos de los períodos de ejecución (inter-arribo de tareas) para el caso de las tareas periódicas.

Las políticas de planificación permiten dar un orden de ejecución a las tareas y garantizar el correcto funcionamiento temporal de las mismas.

El Kernel desarrollado utiliza tres tipos de planificadores:

- FIFO Round Robin.
- Rate Monotonic.
- Earliest Deadline First (EDF).

El planificador FIFO Round Robin es el más simple y se dedica a atender las tareas en base a su prioridad. Las tareas en la cola de listos se encuentran

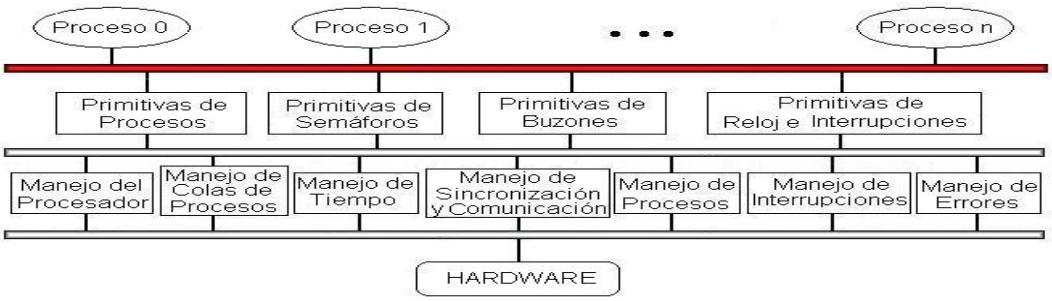

Figura 5: Arquitectura del Kernel

ordenadas de acuerdo a su prioridad, la cual es asignada por el usuario. Las tareas con la misma prioridad son atendidas de forma FIFO Round-Robin.

La planificación Rate Monotonic asigna prioridades a las tareas en base a sus períodos. La tareas con menor período (o mayor frecuencia de ejecución) reciben la mayor prioridad. La asignación de prioridades en la política Rate Monotic es estática, es decir, que se asigna una sola vez, al principio de la ejecución de las tareas, y esta no cambia durante su ejecución. La planificación EDF asigna las prioridades de forma dinámica. La prioridad se asigna de acuerdo a la cercanía de las tareas (de su tiempo de arribo) con respecto a su plazo de respuesta. La tarea con plazo de respuesta más cercano (en un momento determinado) es la que recibe mayor prioridad. Este tipo de planificación permite que las prioridades de las tareas cambien todo el tiempo, dependiendo de su cercanía en ese momento con su plazo de respuesta.

6. Análisis de Planificabilidad para Sistemas de Tiempo Real

La función de un algoritmo de planificación es determinar un orden de ejecución para un conjunto de tareas. Los algoritmos de planificación tienen asociados un análisis de planificabilidad que permite verificar si el conjunto de tareas es factible (si ninguna tarea pierde sus plazos de respuesta). Un sistema en tiempo real se considera factible si, en función del algoritmo de planificación elegido, es capaz de satisfacer todos los tiempos de respuesta de las tareas. Una forma para determinar que el algoritmo de planificación elegido asegura el cumplimiento de los plazos, es realizando un análisis de planificabilidad. Un tipo de análisis consiste en un cálculo de los tiempos de respuesta del peor caso para cada una de las tareas que conforman el sistema. Si los tiempos de respuesta para el peor caso son siempre menores que los plazos de respuesta asociados a las mismas, quiere decir que el sistema es planificable en cualquier condición.

Las características del modelo de las tareas planteadas originalmente para los algoritmos de planificación Rate Monotonic y EDF [5] son las siguientes:

1. Un conjunto de tareas *T* está compuesto por n tareas $\{\tau_1 \dots \tau_i \dots \tau_n\}$. Cada τ_i esta definida de acuerdo a los parámetros: D_i = Plazo, P_i = Período y C_i = Tiempo de ejecución (ejecutado sin interrupción).

2. $C_i \leq D_i = P_i$, es decir, las tareas tienen tiempos de ejecución menores que su plazo, y su plazo es igual a su período.

3. El tiempo de ejecución de cada tarea (C_i) es constante. Siempre se ejecuta a la misma velocidad.

4. Todas las tareas son periódicas.

5. No existen relaciones de precedencia entre las tareas.

6. No se permite la comunicación ni la sincronización entre tareas.

7. No se considera el acceso a recursos compartidos por parte de las tareas.

8. El tiempo de ejecución del cambio de contexto y de las primitivas es despreciable.

9. Todos los procesos se ejecutan en un solo procesador.

6.1. Factor de Utilización (U)

El factor de utilización del procesador para un conjunto dado de tareas representa la fracción de tiempo de procesador empleada en la ejecución de dicho conjunto de tareas.

Para el modelo de sistema presentado, el factor de utilización de una tarea τ_i se expresa como sigue:

$$U_i = \frac{C_i}{T_i} \tag{1}$$

De la ecuación (1) se tiene que el factor de utilización para un conjunto de tareas es:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \quad (2)$$

Es uno de los algoritmos de planificación mas antiguos, sencillo y usado, en este cada proceso tiene asignado un intervalo de tiempo de ejecución llamado *Quantum*. FIFO Round Robin es muy sencillo de implementar, todo lo que necesita el planificador es conocer la cola de los procesos listos y una vez que el proceso en ejecución consume su Quantum, se le quita el procesador y se le asigna al siguiente en la cola, colocando al proceso que salió al final de la cola de procesos listos.

El análisis de planificabilidad para este algoritmo consiste únicamente en verificar si su factor de utilización excede al 100 % ($U \leq 1$).

6.2. Algoritmo Rate Monotonic

En el algoritmo Rate Monotonic, sus autores [5] proporcionan un análisis de planificabilidad de tareas basado en la utilización del procesador. Este análisis, compara la utilización del conjunto de tareas con una cota límite que depende del número de tareas del sistema. En este análisis un conjunto de n tareas no perderá su plazo si cumple con la siguiente condición:

$$U_{TOT} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \quad (3)$$

La tabla 3 describe el comportamiento de la expresión (3) para distintos valores de n. Como se puede apreciar en dicha tabla, al aumentar el número de tareas la utilización mínima garantizada converge en:

$$U_{min} = ln2 \approx 0{,}6931 \quad (4)$$

6.3. Algoritmo FIFO Round Robin

Es uno de los algoritmos de planificación mas antiguos, sencillo y usado, en este cada proceso tiene asignado un intervalo de tiempo de ejecución llamado *Quantum*. FIFO Round Robin es muy sencillo de implementar, todo lo que necesita el planificador es conocer la cola de los procesos listos y una vez que el proceso en ejecución consume su Quantum, se le quita el procesador y se le asigna al siguiente en la cola, colocando al proceso que salió al final de la cola de procesos listos.

El análisis de planificabilidad para este algoritmo consiste únicamente en verificar si su factor de utilización excede al 100 % ($U \leq 1$).

n	U_{min}
1	1
2	0.8284
3	0.7788
4	0.7568
5	0.7334
.	.
.	.
∞	0.6931

Tabla 3: Utilización mínima garantizada para n tareas.

6.4. Algoritmo Earliest Deadline First

El algoritmo de planificación EDF (Earliest Deadline First) es otro algoritmo de planificación para sistemas en tiempo real el cual utiliza el plazo de respuesta de las tareas como la base para asignar prioridades. En esta política de planificación, la tarea con plazo mas cercano es quien recibe la mayor prioridad de ejecución. Bajo este algoritmo, la condición necesaria y suficiente para que un conjunto de tareas periódicas tenga una planificación viable es:

$$U \leq 1 \quad (5)$$

EDF consigue un factor de utilización del 100 % para los conjuntos de tarea que planifica, por lo que se puede decir que es óptimo globalmente, es decir, que si existe un algoritmo que proporcione una planificación factible con un determinado conjunto de tareas periódicas, entonces EDF también proporciona una planificación factible para dicho conjunto de tareas.

7. Ejemplo de Aplicación de Tareas de Tiempo-Real

Como se aprecia en la figura 6 hemos creado una aplicación que consiste de 4 tareas de tiempo real. La prioridad de las tareas es la misma. La tarea 1 representa un reloj en movimiento. Las tareas 2 y 3 ejecutan un juego en el cual un caracter rebota sobre una ventana. Finalmente la tarea 4 consiste en un editor de caracteres que se ejecuta dentro de su ventana. La política de planificación utilizada fue FIFO Round-Robin.

Durante la ejecución de esta aplicación, el Quantum utilizado fue de 5 mili-segundos. Con esta resolución, fue posible observar que cada tarea se ejecuta como si estuviera en un procesador diferente.

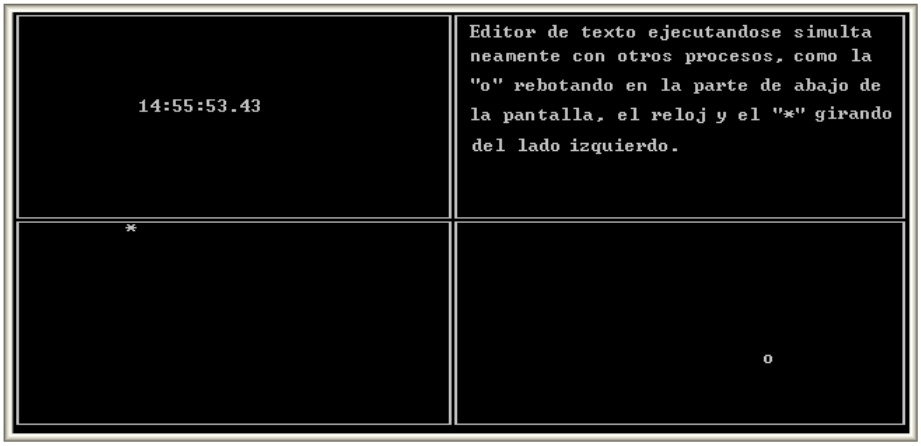

Figura 6: Muestra de 4 tareas ejecutándose sobre el Kernel

8. Ambiente de Desarrollo de Sistemas de Tiempo Real

El Kernel de tiempo real descrito en este artículo, es un componente de un ambiente de desarrollo de sistemas de tiempo real más amplio. El ambiente de desarrollo se muestra en la Figura 7 y consiste de los siguientes componentes:

- **Ambiente Visual para Desarrollo de Sistemas de Tiempo Real**

  Este ambiente visual esta siendo desarrollado junto con el Kernel, y permitirá diseñar los procesos de tiempo real en forma visual utilizando UML y SimuLink. Nuestro objetivo es diseñar sistemas de tiempo real para aplicaciones de control de procesos, en donde pueda ser que no sea necesario programar ni una linea de código.

  Para lograr este objetivo estamos actualmente desarrollando interfaces que permitan que SimuLink (Herramienta para generar aplicaciones de control de procesos) genere el código que pueda ser incluido en el Kernel como una tarea del sistema. Además, dentro de este ambiente visual será posible incluir código en forma visual (de forma parecida a como se realiza en VISUAL C incluyendo componentes de software de tipo general y mediante objetos en UML.

- **Kernel de Tiempo Real Distribuido**.

  El Kernel descrito en este artículo, actualmente esta siendo extendido para un sistema distribuido. Nuestro interés es de que tareas de tiempo real ejecutándose en en distintas computadoras puedan comunicarse y sincronizarse.

- **Ambiente de Simulación**.

  Otro componente de este ambiente de desarrollo es el ambiente de simulación de tareas de tiempo real, el cual permitirá graficar la ejecución de las tareas de tiempo real del sistema en forma estática y dinámica. En el modo estático, será posible definir un conjunto de tareas de tiempo real y graficar su comportamiento. En el modo dinámico, el Kernel se conectará en forma dinámica al ambiente de simulación a fin de presentar en forma gráfica la ejecución del Kernel. Así mismo, el Kernel podrá generar un archivo de texto con su secuencia de ejecución y el ambiente de simulación podrá leer este archivo y graficar su ejecución.

- **Ejecución en Plataformas Embebidas**.

  Actualmente el Kernel de tiempo real se ejecuta sobre MS-DOS, sin embargo, como trabajo futuro nos hemos planteado la necesidad de portar el Kernel en plataformas embebidas. Sobre estas plataformas, será necesario añadir al Kernel un manejador de memoria debido a que actualmente el Kernel utiliza el cargador (loader) de MS-DOS para el asignamiento de memoria y para su ejecución. Además, será necesario el desarrollo de interfaces de aplicación (API) y *drivers* para los dispositivos de entrada/salida requeridos en la plataforma embebida.

9. Conclusiones y Trabajo Futuro

El Kernel descrito en este artículo es capaz de controlar tareas de tiempo real concurrentes a una resolución de un milisegundo. El Kernel contiene distintas primitivas para manejo de procesos, comunicación, sincronización y manejo de tiempo. Así mismo el Kernel desarrollado utiliza los mejores mecanismos de

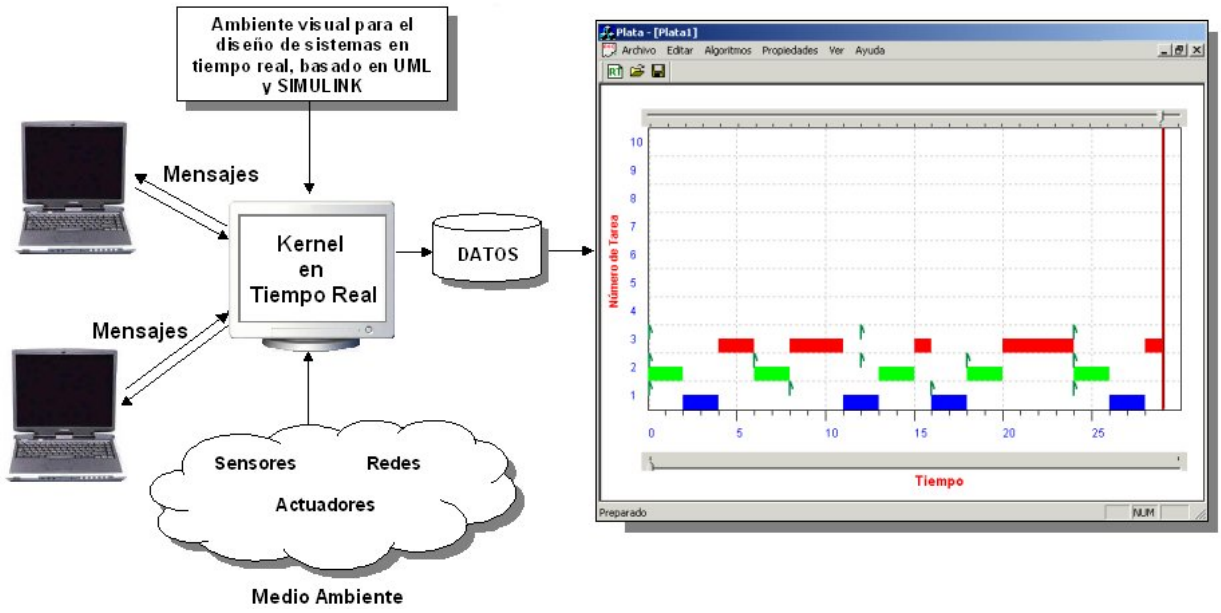

Figura 7: Ambiente de Desarrollo de Sistemas en Tiempo Real Orientado a Control de Procesos

planificación de procesos existentes (Rate Monotonic y EDF) para obtener la mejor eficiencia y menor pérdida de plazos durante la ejecución del mismo.

Como meta final nos proponemos obtener un ambiente de desarrollo de sistemas de tiempo real, en donde sea posible diseñar este tipo de tareas en forma visual. Actualmente nos encontramos en las etapas finales de desarrollo de este ambiente.

Referencias

[1] A. Burns and A. Wellings. *Real-Time Systems and programming languages*, Addison-Wesley, April, 1997.

[2] G. Buttazzo. *Hard Real-Time Computing Systems. Predictable scheduling algorithms and applications.* Kluwer Academic, 1997.

[3] P. Gai, L. Abeni, M. Giorgi, G. Buttazzo. *A New Kernel Approach for Modular Real-Time System Development. Proceedings of the 13th IEEE Euromicro conference on Real-Time Systems, June 2001, Delft, NL.*

[4] J. W. S. Liu. *Real-Time Systems.* Prentice Hall, Primera Edición, Junio 15, 1997.

[5] C. L. Liu and W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the Association for Computing Machinery*, 2:46-61, 1973.

[6] P. Mejía Álvarez. Sistemas operativo en tiempo real para multiples procesos (sopco). *IV Congreso Latinoamericano de Control Automático*, Noviembre, 1990.

[7] P. Mejia Alvarez. Sistemas en Tiempo Real. *Notas del curso de Sistemas en Tiempo Real*, 2001.

[8] K. Gosh, B. Mukherjee, K. Schwan. *A Survey of Real-Time Operating Systems.* Tech Report. GIT-CC-93/18. College of Computing, Georgia Institute of Technology. Feb 1994.

[9] A. Silberschatz and P. B. Galvin. *Operating System Concepts.* Addison Wesley, 1994.

[10] R. Yerraballi. Real-Time operating systems: An ongoing review. *Proceedings of 21st IEEE Real-Time Symposium, WIP sectione Systems, Orlando Florida*, 2000.

[11] K. M. Zuberi and K. G. Shin. Emeralds: A microKernel for embedded real-time systems. *In proceedings of IEEE Real-Time Technology and Applications Symposium PP.241-249*, June , 1996.