

# Sistemas operativos en tiempo real

Enrique Vilches Cruces. Escuela Universitaria Politécnica de Málaga. Curso 2008-2009.

Ingeniería Técnica Industrial Especialidad Mecánica.

## Resumen

Los sistemas operativos en tiempo real fueron diseñados para aplicaciones basadas en el tiempo real, por lo que se exigen continuas correcciones a las acciones o respuestas que da el sistema bajo una restricción de tiempo. En el caso de que dichas restricciones no sean respetadas, diremos que el sistema ha fallado. Para garantizar el correcto funcionamiento de este tipo de sistemas se necesita que éste sea determinista, es decir, que ejecutado con los mismos datos en cualquier otro momento dará la misma respuesta o resultado.

## 1. CARACTERÍSTICAS

1. No se precisa mucha memoria para su ejecución.
2. Fiabilidad.
3. Capacidad de interactuar con el usuario.
4. Cualquier cambio en el hardware puede hacer que se inicialice alguno de sus comandos.
5. Sensibilidad.
6. Tiempo de respuesta predecible para eventos electrónicos.
7. El ya citado determinismo.
8. Multi-arquitectura, es decir, que puede funcionar con distintos tipos de CPU.
9. Tolerancia a los fallos del sistema.

### 1.2. Procesador

Una peculiaridad de estos sistemas operativos es la dificultad de demostrar que cumplen con el período del reloj interno del ordenador para hacer sus ya citadas restricciones, puesto que posee un algoritmo de programación muy especializado y a veces las interrupciones o correcciones que se deben hacer son muchas, poniendo así en compromiso la velocidad de procesamiento de la CPU.

## 2. REQUERIMIENTOS

Existen dos tipos de sistemas operativos de tiempo real en cuanto a las restricciones de las tareas: Los sistemas rígidos de tiempo real en los cuales el incumplimiento de requisitos temporales supone un fallo grave del sistema (Críticos) y los sistemas flexibles de tiempo real en los cuales un incumplimiento esporádico de los requisitos temporales no condiciona al sistema (No críticos). En los sistemas que haremos inciso serán en los que existe una convivencia entre tareas críticas y tareas no críticas.

Una vez esclarecido esto, podemos tratar de los requerimientos del sistema.

Lo fundamental en este tipo de sistemas es la estructura dinámica de tareas de tiempo crítico: Se podrá introducir una nueva tarea crítica siempre y cuando el sistema no se encuentre sobrecargado, para que esta nueva tarea que intentamos insertar no afecte a las demás tareas críticas ya existentes, sin embargo, una tarea no crítica siempre podrá ser añadida, pues solo basta la ralentización de su procesamiento para no afectar a las tareas críticas ya existentes.

En aplicaciones de tiempo real lo importante es realizar sus tareas justo en el momento adecuado, ni antes ni después, es decir, en instantes fijos y predeterminados (Determinismo).

### **3. PROGRAMACIÓN.**

En los diseños más comunes de programación cada tarea tiene tres estados: ejecución, preparada y bloqueada, aunque la mayoría de las tareas están bloqueadas casi todo el tiempo. Sólo se ejecuta una tarea por CPU y la lista de las tareas suele ser muy corta, de dos o tres tareas como mucho.

Lo más complicado de este tipo de sistemas operativos es diseñar el programador. Normalmente la estructura de los datos de la lista de tareas del programador es diseñada para que necesiten interrupciones de tiempo solamente durante períodos de tiempo muy cortos, cuando se busca una parte de la lista muy específica.

Esto último da lugar a que otras tareas de la lista puedan ser ejecutadas independientemente del tiempo de interrupción de alguna otra tarea, son asíncronas.

Un ejemplo de buena programación es una lista conectada de tareas ordenadas por prioridad. Como la mayoría de las listas de tareas solo disponen de dos o tres comandos, una búsqueda secuencial siempre va a ser la respuesta más rápida, puesto que necesita de un tiempo de ejecución menor que el resto de programaciones.

El tiempo que se necesita para poner en cola una nueva tarea y reconfigurar el estado de la tarea con la prioridad más alta se llama “tiempo de respuesta crítico”.

Preparar una nueva tarea precisa de 3 a 20 instrucciones por cada entrada en la cola y restaurar la tarea de prioridad máxima necesita de 5 a 30 instrucciones.

#### **3.1. Sistema de interacción entre tareas.**

Existía un pequeño problema que escapa a la programación de estos sistemas, y era el que las distintas tareas en algunas ocasiones debían utilizar los mismos datos o componentes físicos al mismo tiempo, pero no era computable que fueran a la vez. Para solucionar este inconveniente se ideó una especie de código donde las tareas interactuaban entre sí, se idearon dos modelos.

El primero que mencionaremos es el sistema de semáforos, en el cual, cuando un semáforo estaba cerrado había una cola de instrucciones esperando la apertura de éste. No obstante surgieron dos problemas a este modelo: “inversión de propiedades” y “puntos muertos”.

En la inversión de propiedades nos encontramos con que una tarea de máxima prioridad no se ejecuta porque otra de mínima prioridad tiene un semáforo. Si se diera el caso en el que una tarea de prioridad intermedia no dejase ejecutar la de prioridad baja, la de prioridad máxima nunca se ejecutaría.

En los puntos muertos se da el caso en el que dos tareas tienen dos semáforos pero en orden inverso. Esto se resuelve normalmente mediante un diseño cuidadoso, realizando colas o quitando semáforos, que pasan el control de un semáforo a la tarea de más alta prioridad en determinadas condiciones.

El otro tipo de interacción entre tareas es que se manden mensajes entre ellas. No obstante se sigue teniendo el mismo problema de inversión de prioridades que tiene lugar cuando una tarea está funcionando en un mensaje de baja prioridad, e ignora un mensaje de más alta prioridad en su correo. Los puntos muertos ocurren cuando dos tareas esperan a que la otra responda.

Está comprobado que el sistema de mensajes entre tareas da mucho mejores resultados y mejor rendimiento que el sistema de semáforos para sistemas en tiempo real.

### 3.2. Las interrupciones.

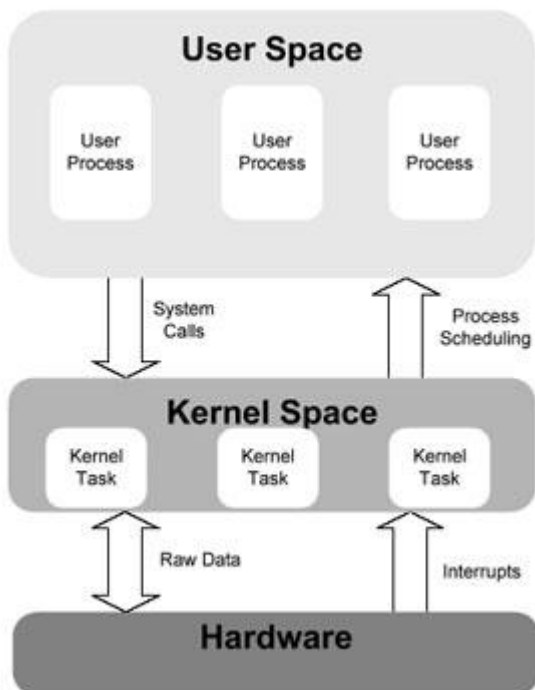
En este tipo de sistemas, las interrupciones es la forma más común de aportar información del exterior del programa. En un sistema de tiempo real, una interrupción puede significar desde la existencia de nueva información en un puerto de comunicaciones hasta una muestra de audio-video obtenida por una webcam.

Para que el sistema este acorde con su nombre y que sea en tiempo real, debe procesar una interrupción obtenida antes de que se cumpla la siguiente. Como el microprocesador normalmente solo puede atender una interrupción a la vez, es necesario que los controladores de tiempo real se ejecuten en el menor tiempo posible. Esto se logra no procesando la señal dentro de la interrupción, sino enviando un mensaje a una tarea o solucionando un semáforo que está siendo esperado por una tarea. El programador se encarga de activar la tarea y esta se encarga de adquirir la información y completar el procesamiento de la misma.

El tiempo que transcurre entre la generación de la interrupción y el momento en el cual esta es atendida se llama latencia de interrupción. El inverso de esta latencia es una frecuencia llamada frecuencia de saturación, si las señales que están siendo procesadas tienen una frecuencia mayor a la de saturación, el sistema será físicamente incapaz de procesarlas. En todo caso la mayor frecuencia que puede procesarse es mucho menor que la frecuencia de saturación y depende de las operaciones que deban realizarse sobre la información recibida.

### 4. ARQUITECTURA DEL SISTEMA.

La memoria física de un ordenador está dividida en dos partes, una reservada para el usuario “User space” y otra reservada para el kernel “kernel space”. El kernel multitarea es capaz de manejar múltiples aplicaciones del usuario, que como tales se ejecutan en el espacio de la memoria reservado para dicho fin, y les hace creer que disponen de todo el espacio de la memoria física y de absolutamente todos el hardware. Las operaciones entre los programas que se encuentran en el espacio del usuario y el espacio kernel se realizan mediante las llamadas al sistema. Éstas llamadas lo que hacen es acceder a recursos físicos compartidos por las dos partes de la memoria. El kernel controla el acceso al hardware del sistema y por ello los programas del espacio de memoria destinado al usuario no conocen los detalles físicos de estas maquinas.



**Figura 1: Arquitectura de un sistema operativo en tiempo real.**

En la Figura 1 tenemos el ejemplo de arquitectura de este tipo de sistemas operativos.

Observamos que el espacio de memoria física está dividido en dos partes, una para el usuario y otra para el kernel. También podemos observar que el hardware solo interactúa con el kernel y nunca con el espacio de usuario, las interrupciones llegan del hardware al kernel y el kernel lo comunica al usuario.

Una medida ya comentada anteriormente del rendimiento de este tipo de sistema operativo es la latencia que es el tiempo desde que ocurre un evento hasta que es tratado por el sistema, pero la otra medida es el jitter, que son las variaciones del periodo normal de los eventos ya citados.

La mayoría de los sistemas operativos tienden a tener una latencia y jitter bajos, pero un sistema operativo en tiempo real necesita que estos dos parámetros estén definidos y que a su vez no estén directamente relacionados con la carga del sistema.

## 5. PROBLEMAS CON LA MEMORIA.

Con este tipo de sistemas operativos nos encontramos con dos tipos de problemas en cuanto al reparto de la memoria.

El primero, la velocidad del reparto es importante. Un esquema de reparto de memoria estándar recorre una lista conectada de longitud indeterminada para encontrar un bloque de memoria libre; sin embargo, esto no es aceptable ya que el reparto de la memoria debe ocurrir en un tiempo fijo en el sistema operativo.

En segundo lugar, la memoria puede fragmentarse cuando las regiones libres se pueden separar por regiones que están en uso. Esto puede provocar que se pare un programa, sin posibilidad de obtener memoria, aunque en teoría exista suficiente memoria. Una solución es tener una lista vinculada LIFO de bloques de memoria de tamaño fijo. El término LIFO alude a que guarda analogía con una pila de platos, en la que los platos van poniéndose uno sobre el otro, y si se quiere sacar uno, se saca primero el último que se puso. Esto funciona asombrosamente bien en un sistema simple.

Ahora bien, la paginación se desactiva en estos sistemas operativos, puesto que es un factor bastante aleatorio e impredecible, lo cual nos induciría a que el sistema no cumpliera con los períodos de interrupción como es debido.

## 6. EJEMPLOS DE SISTEMAS OPERATIVOS EN TIEMPO REAL.

- QNX
- LynxOS
- Redhat Embedded Linux
- SOOS
- Ubuntu Studio (Linux)
- VxWorks
- Windows CE
- Linchos
- UNIX

## 7. CONCLUSIONES.

Este tipo de sistemas está muy presente en nuestras vidas, pues está pensando como un método de interacción entre usuario y máquina, dispuesto a satisfacer las necesidades de cada usuario.

Estos sistemas están presentes en dispositivos desde teléfonos móviles hasta aviones y aeronaves, por lo que disponen de un alto porcentaje de fiabilidad y de manejabilidad.

### Referencias

[1] <http://www.monografias.com/trabajos37/sistemas-tiempo-real/sistemas-tiempo-real2.shtml>

[2] [http://es.wikipedia.org/wiki/Sistema\\_operativo\\_de\\_tiempo\\_real](http://es.wikipedia.org/wiki/Sistema_operativo_de_tiempo_real)

[3] <http://gayuba1.datsi.fi.upm.es/~dlopez/rtos.php>

[4] <http://biblioteca.universia.net/ficha.do?id=38560260>

[5] <http://www.pablin.com.ar/electron/info/portos/>

[6] <http://wichar.blogspot.com/2009/02/sistemas-operativos-en-tiempo-real.html>

[7] <http://www-gti.det.uvigo.es/~pedro/pub/sodtr/>