Preparador Informática

www.preparadorinformatica.com

TEMA 28. INFORMÁTICA

PROGRAMACIÓN EN TIEMPO REAL. INTERRUPCIONES. SINCRONIZACIÓN Y COMUNICACIÓN ENTRE TAREAS. LENGUAJES

TEMA 30. SAI

PROGRAMACIÓN EN TIEMPO REAL. INTERRUPCIONES. SINCRONIZACIÓN Y COMUNICACIÓN ENTRE TAREAS.

TEMA 28 INF: PROGRAMACIÓN EN TIEMPO REAL. INTERRUPCIONES, SINCRONIZACIÓN Y COMUNICACIÓN ENTRE TAREAS. LENGUAJES

TEMA 30 SAI: PROGRAMACIÓN EN TIEMPO REAL. INTERRUPCIONES, SINCRONIZACIÓN Y COMUNICACIÓN ENTRE TAREAS.

1. INTRODUCCION

En los sistemas informáticos tradicionales se garantiza que ante determinadas entradas se generan las salidas adecuadas. Se tiene en cuenta únicamente sus interacciones con elementos externos (usuarios, dispositivos de entrada...) que puedan producir cambios de estado en el sistema. Entre sus requisitos se pueden incluir restricciones temporales, pero éstas en ningún caso determinarán si el funcionamiento del sistema es o no correcto.

Existen otros entornos en los que las restricciones de tiempo son muy rigurosas, y una respuesta correcta que se produzca fuera de tiempo pasa a ser una respuesta incorrecta. Para dar respuesta a estos entornos surge la programación en tiempo real y los sistemas en tiempo real. Los sistemas en tiempo real están muy acoplados en el mundo externo. Deben responder al ámbito del problema en un tiempo dictado por el ámbito del problema. Debido a que el software de tiempo real debe operar bajo restricciones de rendimiento muy rigurosas, el diseño del software está conducido frecuentemente, tanto por la arquitectura del hardware como por la del software, por las características del sistema operativo, por los requisitos de aplicación y tanto por los extras del lenguaje de programación como por aspectos de diseño. De los lenguajes y la programación en tiempo real es sobre lo que trata este tema.

2. SISTEMAS DE TIEMPO REAL

Un sistema de tiempo real es aquel que debe responder correctamente a una entrada externa en un período especificado de tiempo, el cual debe ser suficiente para resolver un determinado problema. Si las restricciones de tiempo no son respetadas se dice que el sistema ha fallado.

El rendimiento de un sistema de tiempo real se determina principalmente por:

- La velocidad de transferencia: Indica la rapidez de los datos para entrar o salir del sistema.
- El tiempo de respuesta: Es el tiempo que transcurre entre la detección de un suceso interno o externo y la respuesta con una acción. Los parámetros clave que le afectan son:
    - El cambio de contexto: Tiempo y sobrecarga necesarios para conmutar entre tareas.

- La latencia de interrupción: Tiempo que pasa antes de que el cambio sea posible.

2.1 CARACTERÍSTICAS

- **Fiabilidad y seguridad:** El sistema no sólo debe estar libre de fallos, sino que los servicios que presta no deben degradarse más allá de un límite determinado. Deben proporcionar mecanismos específicos para la gestión del tiempo, que permitan realizar las tareas en un tiempo establecido, ordenarlas por prioridad o administrar el tiempo de proceso de cada una, con el fin de garantizar que los resultados ofrecidos por el sistema sean siempre correctos tanto en su funcionalidad como en su tiempo de respuesta.

- **Determinismo:** Es la capacidad de determinar con una alta probabilidad cuál es el tiempo que tarda una tarea en iniciarse. Importantísimo en sistemas que necesitan que ciertas tareas se ejecuten antes de que otras puedan iniciar.

- **Interacción con el entorno:** Los sistemas de tiempo real interactúan con un entorno externo por lo que es necesario utilizar sensores que permitan realizar la toma de datos del entorno y un conjunto de actuadores que permitan modificar el estado del sistema controlado.

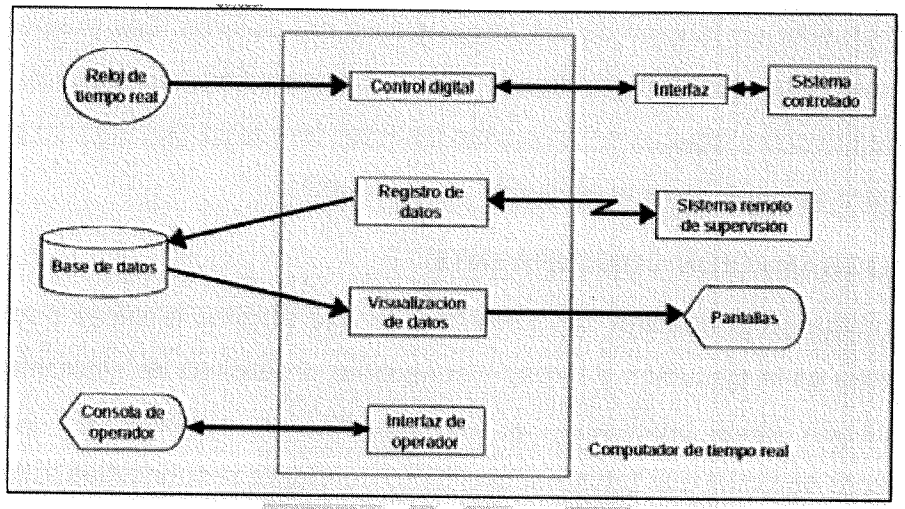
- **Robustez:** Deben estar diseñados para trabajar en cualquier escenario. Un sistema es robusto si tiene la capacidad necesaria para trabajar en condiciones de carga elevada sin colapsarse, es capaz de notificar por adelantado que una tarea no se puede realizar en el plazo establecido para poder tomar acciones alternativas a tiempo, y debe garantizar que un fallo de hardware o software no afecte al rendimiento del sistema (tolerancia a fallos).

- **Concurrencia:** Las tareas de un sistema de tiempo real deben poder ejecutarse en paralelo, sin necesidad de esperar a que una termine para la ejecución de otra tarea. El software se encargará de gestionar la ejecución en paralelo y la sincronización de las tareas en caso de ser necesario.

- **Recuperación de fallos:** Cuando ocurra un fallo, el sistema debe preservar la mayor parte de los datos y capacidades, incorporando redundancias para asegurar la reinicialización del sistema.

2.2. FUNCIONAMIENTO

La estructura general de funcionamiento de un sistema en tiempo real se puede resumir en el siguiente gráfico:



El funcionamiento de un sistema en tiempo real se basará en la ejecución de dos tipos de tareas y actividades:

- Tarea de control: La parte principal de un sistema de tiempo real, su función es la de supervisar y actuar sobre el sistema de tiempo real. Estas tareas serán llevadas a cabo por los sensores y actuadores.
- Tarea de ejecución: Tareas ejecutadas en respuesta a los eventos detectados por el sistema de control. El modelo software será el encargado de comprobar los valores proporcionados por sensores y actuadores y actuar en función de los mismo.

2.3. APLICACIONES

Los sistemas de tiempo real pueden tener muchísimas y con el paso del tiempo y el desarrollo de nuevas tecnologías surgen nuevos campos de utilización para estos sistemas. Las áreas más comunes donde se aplican los servicios de un STR podrían ser las telecomunicaciones, los sistemas multimedia, el control industrial, la robótica, los sistemas de aviónica y espaciales, los ferrocarriles, los automóviles, electrodomésticos de nueva generación, experimentos científicos y sistemas médicos.

3. PROGRAMACIÓN EN TIEMPO REAL

La programación de sistemas de tiempo real es un tipo de programación por la cual la secuencia de algunas de las acciones se determina por los sucesos externos al programa. Estas acciones se dividen en procesos que se ejecutan asíncronamente en función de las necesidades del sistema. Los programas en tiempo real están fuertemente ligados a la programación concurrente, que se puede definir como aquella en la que las distintas tareas se ejecutan en paralelo, pero dentro de cada tarea la ejecución es secuencial.

3.1. PROGRAMACIÓN CONCURRENTE

Se denomina programación concurrente a la que emplea técnicas para lograr paralelismo en la ejecución d tareas y solucionar problemas de sincronización y comunicación entre procesos que se ejecutan en paralelo. Un programa concurrente es aquel que presenta una serie de tareas secuenciales autónomas que pueden ejecutarse en paralelo. Todos los lenguajes de programación concurrente tienen un elemento tarea, que tiene un único hilo de ejecución. La ejecución de estas tareas puede ocurrir de diferentes formas:

- En un único procesador, multiprogramación.
- En un sistema multiprocesador con memoria compartida, multiproceso.
- En un sistema multiprocesador sin memoria compartida, sistema distribuido.

Es el algoritmo usado para la planificación de tareas quien se encarga de determinar el orden de ejecución de las mismas, que no debe afectar al resultado final de la ejecución.

3.2. REPRESENTACIÓN DE LOS PROCESOS CONCURRENTES

Existen varios mecanismos básicos de representación de la ejecución de procesos concurrentes.

a) **Corrutinas:** Las corrutinas son como subrutinas con la diferencia que entre ellas se pueden pasar el control de una manera simétrica, en lugar de una manera jerárquica. El control se pasa de una corrutina a otra con una sentencia *resume*, que significa que se reanude la ejecución de una corrutina en el lugar donde había suspendido su ejecución, suspendiéndose la corrutina

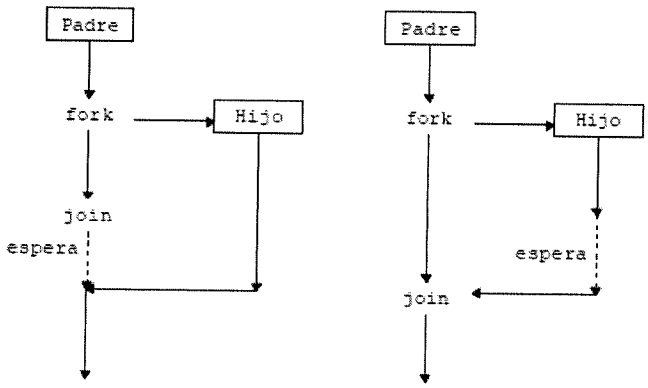
actual hasta que sea activada por otra. El lenguaje C permite el uso de corrutinas con las funciones *setjmp y longjmp*.

b) Fork y Join: La sentencia fork especifica que una determinada rutina comenzará a ejecutarse de forma concurrente con quien invoca a fork. La sentencia join permite que el invocador se sincronice con la finalización de la rutina que ha lanzado, esperando a que ella termine si no lo ha hecho.

```
function F return ...
..
end F;
procedure P
        ...
        C := fork F;
        ...
        J := join C;
end P
```

Entre la ejecución del fork y del join del procedimiento P, la función F se ejecuta de forma concurrente.

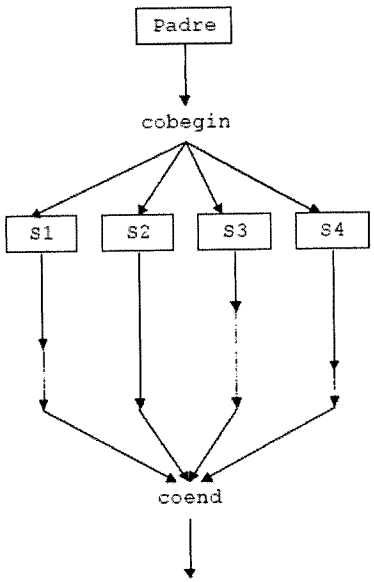
fork y join permiten la creación dinámica de procesos y ofrece un mecanismo para pasar información al proceso hijo a través de parámetros. Normalmente, la función join solo permite devolver un único parámetro usado para indicar el estado de error al finalizar el proceso hijo.



c) Cobegin y Coend: El cobegin (o copar o par) es una forma estructurada de indicar la ejecución concurrente de un grupo de sentencias:

```
cobegin
        S1;
        S2;
        S2;
        .
        .
        Sn;
coend
```

Esta construcción hace que todas las sentencias S1, S2 ... se ejecuten concurrentemente. La sentencia cobegin finaliza sólo cuando han terminado todas las sentencias.



Las sentencias Sn pueden ser cualquier construcción que permita el lenguaje, desde simples asignaciones hasta llamadas a subprogramas. Si se emplean llamadas a subprogramas se les puede pasar información a los nuevos procesos utilizando el paso de parámetros. Una de estas sentencias podría contener a su vez otra construcción cobegin/coend, creándose una jerarquía de procesos.

d) Declaración explícita.

Aunque varias rutinas secuenciales se pueden ejecutar concurrentemente utilizando las construcciones fork/join y cobegin/coend, la estructura de un programa puede quedar más clara si las rutinas se declaran ellas mismas como concurrentes. La declaración explícita de procesos ofrece esta posibilidad. Así por ejemplo Java utiliza la declaración de métodos concurrentes mediante la interfaz runnable.

3.3. PLANIFICACIÓN DE LA EJECUCIÓN

Se entiende como planificación de la ejecución el determinar qué proceso debe ejecutarse en cada instante de tiempo, con la finalidad de que el sistema cumpla con los requisitos temporales.

Entre los sistemas operativos actuales se emplean una serie de interrupciones de reloj periódicas, de modo que en cada interrupción de reloj el sistema recupera el control y decide cuál será el proceso que pasará a ejecutarse, impidiendo de esta forma que un proceso acapare la CPU. A este tipo de programación se le denomina programación expulsiva y es necesaria en sistemas de tiempo real.

4. INTERRUPCIONES

Una interrupción es un mecanismo con el que se puede detener temporalmente el flujo normal del programa en ejecución, al que debe responder el sistema en un tiempo finito y especificado. Cuando se produce una interrupción, el flujo normal de procesamiento es modificado por un suceso que necesita un servicio inmediato.

Con frecuencia se presentan **interrupciones múltiples**, por lo que se deben establecer prioridades e interrupciones multinivel, de tal forma que la tarea con mayor prioridad se ejecute dentro de las restricciones de tiempo especificadas e independientemente de otros sucesos.

4.1. TRATAMIENTO DE INTERRUPCIONES

Después de cada instrucción la CPU verifica la línea de interrupción. Si se encuentra activa indica que se ha producido una interrupción, en caso contrario se pasa a la siguiente instrucción y se repite el ciclo. Las IRQ (Interrupt ReQuest) son líneas que llegan al controlador de interrupciones, un componente hardware dedicado a la gestión de las interrupciones, y que puede estar integrado en la CPU o ser un circuito separado conectado a la CPU.

El controlador de interrupciones debe ser capaz de habilitar o inhibir líneas de interrupción, y establecer prioridades entre las distintas interrupciones habilitadas. Cuando varias líneas de petición de interrupción se activan a la vez, el controlador de interrupciones utilizará estas prioridades para escoger la interrupción sobre la que informará al procesador principal. Sin embargo, hay interrupciones que no se pueden deshabilitar y son llamadas interrupciones no enmascarables o NMI (Non Maskable Interrupt).

Un procesador principal (sin controlador de interrupciones integrado) suele tener una única línea de interrupción llamada habitualmente INT. Esta línea es activada por el controlador de interrupciones cuando tiene una interrupción que servir. Al activarse esta línea, el procesador completa la ejecución de la instrucción en curso y guarda el estado del programa en la pila.

Después el procesador consulta los registros del controlador de interrupciones para averiguar qué IRQ es la que ha de atender. A partir del número de IRQ

busca en el vector de interrupciones qué rutina debe llamar para atender la petición del dispositivo asociado a dicha IRQ.

El vector de interrupciones es un vector que contiene el valor que apunta a la dirección en memoria de la rutina servidora de interrupción. En muchas arquitecturas de ordenadores los vectores de interrupción se almacenan en una tabla en una zona de memoria, de modo que cuando se atiende una petición de interrupción de número n, el sistema transfiere el control a la dirección indicada por el elemento n-ésimo de dicha tabla.

Otras maneras de ejecutar el gestor de la interrupción son las siguientes:

- Cargar el contador de programa con un nuevo valor desde un registro específico o desde una posición de memoria.
- Ejecutar la instrucción de llamada en una dirección proporcionada por un sistema externo.
- Utilizar una señal de salida para reconocer la interrupción y tomar la instrucción de un dispositivo externo.

Una vez finalizada la rutina servidora de interrupción, el procesador restaura el estado del programa interrumpido y vuelve al punto anterior a la interrupción.

5. SINCRONIZACIÓN Y COMUNICACIÓN ENTRE TAREAS

En los sistemas multitarea, los procesos comparten un conjunto de recursos que pueden ser datos almacenados en la memoria o dispositivos de entrada/salida. Sin embargo, el acceso concurrente a un recurso puede hacer que la acción de un proceso interfiera en las acciones de otro de una forma no adecuada. Esto es debido a que una instrucción sencilla se descompone en operaciones elementales, y se puede dar la situación de que el planificador de procesos permita el entrelazado de las operaciones elementales de cada uno de los procesos, lo que inevitablemente producirá errores. Para evitar este tipo de errores se pueden identificar aquellas regiones de los procesos que acceden a recursos compartidos, y ejecutarlas como si fueran una única instrucción.

Se denomina **sección crítica** a aquellas partes de los procesos que no pueden ejecutarse de forma concurrente, y que desde otro proceso se ven como si fueran una única instrucción. Esto quiere decir que, si un proceso entra a ejecutar una

sección crítica en la que se accede a unos recursos compartidos, otro proceso no puede entrar a ejecutar la misma sección crítica. Las secciones críticas se pueden agrupar en clases, siendo mutuamente exclusivas las secciones críticas de cada clase. Para conseguir dicha exclusión se deben implementar protocolos software que bloqueen el acceso a una sección crítica mientras está siendo utilizada por otro proceso.

La **sincronización** es el mecanismo que controla el orden en que se ejecutan las tareas para cumplir las restricciones en la intercalación de las operaciones de cada una de ellas. Puede ser de cooperación, cuando una tarea debe esperar a que otra finalice alguna actividad para poder continuar su ejecución; o de competición, cuando una tarea debe esperar a que otra finalice la utilización de algún recurso compartido para poder continuar su ejecución.

La **comunicación** es un mecanismo de transferencia de información de un proceso a otro que puede requerir o no de sincronización entre ellos.

5.1. SEÑALES

Es un mecanismo de comunicación asíncrono equivalente a las interrupciones producido por el sistema operativo. Sirven para el manejo de excepciones, la notificación de eventos asíncronos, la terminación anormal de tareas y la comunicación explícita entre procesos (kill).

Las señales no siempre se atienden justo después de su envío ya que la tarea receptora puede estar suspendida. Una tarea puede definir cómo manejar cada señal que recibe, ya sea ignorándola, ejecutando la acción por defecto asociada a la señal, o cambiando su estado de ejecución. Cada tipo de señal tiene asignado un identificador y a cada identificador le corresponde un número entero que identifica de firma unívoca cada una de las señales.

5.2. SEMÁFOROS

Un semáforo es un objeto de datos compartido por varios procesos, y está formado por:

- Un indicador S que se inicializa con el número total de recursos similares sobre los que se desea sincronizar tareas. Si S > 0 entonces existen recursos

disponibles y la tarea lo puede utilizar; si S = 0 no hay recursos disponibles y el proceso debe esperar.

- Una cola de tareas a la cual se añaden los procesos en espera. Un proceso en espera de un semáforo no está en ejecución ni listo para pasar a ejecución, puesto que no tiene la CPU ni puede pasar a tenerla mientras que no se lo indique el semáforo.

Sólo se permiten tres operaciones sobre un semáforo: inicializa, espera y señal.

La exclusión mutua se realiza fácilmente utilizando semáforos. La operación de espera se usará como procedimiento de bloqueo antes de acceder a una sección crítica y la operación señal como procedimiento de desbloqueo, de tal manera que el proceso que ejecuta la operación de espera queda bloqueado hasta que el otro proceso ejecuta la operación de señal. Se utilizarán tantos semáforos como clases de secciones críticas se establezcan.

Las operaciones del semáforo son procedimientos que se implementan como acciones indivisibles, y por ello la comprobación y el cambio de valor del indicador se efectúan de manera real como una sola operación. El problema es que, si una tarea no realiza la operación señal en el punto apropiado, el sistema completo de tareas se puede bloquear.

5.3. MONITORES

Un monitor es un conjunto de procedimientos encapsulados dentro de un módulo, que proporciona el acceso con exclusión mutua a un conjunto de recursos compartidos por un grupo de procesos. Tiene la propiedad especial de que sólo un proceso puede estar activo cada vez para ejecutar un procedimiento del monitor, los demás deben permanecer en espera.

En este caso la exclusión mutua está implícita. La única acción que debe realizar el programador del proceso que usa un recurso es invocar una entrada del monitor, al contrario que en los semáforos, donde el programador debe proporcionar la correcta secuencia de operaciones espera y señal para no bloquear al sistema.

El mecanismo de sincronización de un monitor funciona de la siguiente manera:

- Se asocia una variable de condición a cada recurso por el que un proceso deba esperar.
- Cuando un proceso ejecuta una operación de espera, se suspende y se coloca en una cola asociada a dicha variable de condición. La suspensión del proceso hace que se libere la posesión del monitor, y permite que entre otro proceso.
- Cuando un proceso ejecuta una operación de señal, se libera un proceso suspendido en la cola de la variable de condición utilizada. Si no hay ningún proceso suspendido en la cola de la variable de condición invocada, la operación señal no tiene ningún efecto.
- El monitor debe usar un sistema de prioridades en la asignación del recurso que impida que un proceso en espera se vea postergado indefinidamente por otros procesos nuevos.

5.4. MENSAJES

Un mensaje es una transferencia de información de una tarea a otra. Proporciona un medio para que cada tarea sincronice sus acciones con otra tarea, y sin embargo la tarea permanece libre para continuar ejecutándose cuando no necesita estar sincronizada. Los mensajes integran de manera inseparable la sincronización y la comunicación entre procesos. La comunicación mediante mensajes necesita siempre de un proceso emisor, de un proceso receptor, de un enlace entre ambos y de información que intercambiarse.

Las operaciones básicas son enviar(mensaje) y recibir(mensaje). Los sistemas operativos tienen asociado a cada enlace una cola en la cual mantienen los mensajes pendientes de ser recibidos.

Existen dos maneras de referirse a los procesos emisor y receptor en un mensaje:

- Directa. Ambos procesos, el que envía y el que recibe, nombran de forma explícita al proceso con el que se comunican. Las operaciones de enviar y recibir toman la forma:

  enviar(Q, mensaje): envía un mensaje al proceso Q.

  recibir(P, mensaje): recibe un mensaje del proceso P.

Este método de comunicación establece un enlace entre dos procesos que desean comunicarse, proporcionando seguridad en el intercambio de mensajes, ya que cada proceso debe conocer la identidad de su pareja en la comunicación, pero por lo mismo no resulta muy adecuado para implementar rutinas de servicio de un sistema operativo.

- Indirecta. Los mensajes se envían y reciben a través de una entidad intermedia en el que los procesos dejan mensajes y del cual pueden ser tomados por otros procesos, denominada buzón. Cada buzón tiene un identificador que lo distingue. En este caso las operaciones básicas de comunicación toman la forma:

  enviar(buzónA, mensaje): envía un mensaje al buzón A.

  recibir(buzónA, mensaje): recibe un mensaje del buzón A.

La sincronización entre los procesos implicados en el mensaje puede ser por:

- Envío síncrono. Necesita que el emisor y el receptor se encuentren para realizar una comunicación, es decir, que estén en el mismo punto de ejecución.
- Envío asíncrono. El proceso que envía el mensaje sigue su ejecución sin preocuparse de la recepción del mismo. El sistema operativo se encarga de recoger el mensaje del emisor y almacenarlo en espera de que una operación de recibir lo recoja.
- Invocación remota. El proceso que envía el mensaje sólo prosigue su ejecución cuando ha recibido una respuesta (recibido o error) del receptor.

El intercambio de información se puede realizar de dos formas:

- Por valor. Se realiza una copia del mensaje del espacio de direcciones del emisor al espacio de direcciones del receptor. Tiene la ventaja de que mantiene el desacoplo en la información que maneja el emisor y el receptor, lo que proporciona mayor seguridad en la integridad de la información. Tiene el inconveniente del gasto de memoria y tiempo que implica la copia.
- Por referencia. Se pasa un puntero al mensaje. Tiene la ventaja del ahorro de memoria y tiempo, y el inconveniente de necesitar mecanismos adicionales de seguridad para compartir la información entre los procesos.

6. LENGUAJES

Para aplicaciones en tiempo real pueden usarse con efectividad muchos lenguajes de programación de propósito general (C, FORTRAN, ...). Sin embargo, una clase llamada lenguajes de tiempo real se utilizan frecuentemente para estas aplicaciones especializadas (Ada, Jovial, HAL/S, Chill...)

Es importante que el lenguaje de programación soporte directamente la multitarea debido a que los sistemas de tiempo real deben responder a sucesos asíncronos que ocurren simultáneamente. Aunque muchos sistemas operativos de tiempo real dan capacidad de multitarea, frecuentemente existe software de tiempo real suficiente para la ejecución del programa en tiempo real. El soporte de tiempo de ejecución requiere menos memoria que un sistema operativo y puede ser adaptado a una aplicación, incrementando así el rendimiento.

Un sistema de tiempo real que haya sido diseñado para acomodar múltiples tareas debe también acomodar la sincronización entre tareas. Un lenguaje de programación que soporte directamente primitivas de sincronización, tales como SCHEDULE, SIGNAL y WAIT, simplifica mucho la traducción del diseño al código. La orden SCHEDULE planifica un proceso basándose en el tiempo o en un suceso; las órdenes SIGNAL y WAIT manipulan un semáforo, que facilita la sincronización de tareas concurrentes.

Finalmente, son necesarias facilidades que permitan una programación fiable, debido a que los programas de tiempo real son frecuentemente grandes y complejos. Estas características incluyen la programación modular, un fuerte reforzamiento de la tipificación de los datos y muchas otras construcciones de control y definición de datos.

7. CONCLUSIÓN

La evolución en el mercado de los procesadores y de los sistemas de procesamiento en general gira en torno a la integración de más unidades físicas de procesamiento que permitan ejecutar una mayor cantidad de tareas de manera simultánea. Una de las principales consecuencias de este planteamiento, desde el punto de vista de la programación, es la necesidad de

utilizar herramientas que permitan gestionar adecuadamente el acceso concurrente a recursos y datos por parte de distintos procesos o hilos de ejecución, entre otros aspectos.

8. BIBLIOGRAFÍA

- Prieto, A. y otros. **Introducción a la informática**. Editorial McGraw-Hill.
- López, J. **Programación en tiempo real y bases de datos**: Un enfoque práctico. Editorial UPCGRAU
- Vallejo, D. y otros. **Programación Concurrente y Tiempo Real**. Editorial Edlibrix
- http://atc.ugr.es/APrieto_videoclases Departamento de Arquitectura y Tecnología de Computadores. Universidad de Granada.