

Programación en tiempo real.  
Interrupciones. Sincronización  
y comunicación entre tareas.  
Lenguajes.

## **TEMA 28 (30 SAI)**

---

**ABACUS NT**

## **Índice**

---

- 1. Introducción.**
- 2. Sistemas en tiempo real**
  - 2.1. Aspectos de integración y de rendimiento**
  - 2.2. Sistemas operativos de tiempo real (SOTR)**
  - 2.3. Características de los sistemas de tiempo real.**
  - 2.4. Clasificación de los sistemas de tiempo real.**
  - 2.5. Fiabilidad y tolerancia a fallos**
    - 2.5.1. Técnicas de programación para la tolerancia a fallos
  - 2.6. Sincronización con sucesos externos**
- 3. Manejo de interrupciones**
  - 3.1. Concepto. Interrupciones múltiples.**
  - 3.2. Tratamiento de interrupciones.**
- 4. Sincronización y comunicación de tareas**
  - 4.1. Semáforos**
  - 4.2. Memoria Compartida**
  - 4.3. Intercambio de mensajes**
  - 4.4. Buzones y puertos**
  - 4.5. Conductos (pipes)**
  - 4.6. Llamadas a procedimientos remotos (RPC)**
  - 4.7. Bases de datos de tiempo real**
- 5. Lenguajes de tiempo real**
  - 5.1. Lenguajes utilizados para la programación en tiempo Real**
    - 5.1.1. ADA
    - 5.1.2. JAVA
    - 5.1.3. C
- 6. Conclusión**
  - 6.1. Relación del tema con el sistema educativo actual**
- 7. Bibliografía**

## 1. Introducción.

El software de tiempo real está muy acoplado en el mundo externo. Debe responder al ámbito del problema (el mundo real) 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.

Como cualquier sistema basado en computadora, un sistema de tiempo real debe integrar hardware, software, recursos humanos y elementos de una base de datos, para conseguir adecuadamente un conjunto de requisitos funcionales y de rendimiento.

El problema para los sistemas de tiempo real es realizar la asignación adecuada de funciones y comportamiento a los elementos del sistema. El rendimiento en tiempo real es frecuentemente tan importante como la función, pero las decisiones de asignación relativas al rendimiento son, frecuentemente, difíciles de hacer con seguridad. ¿Puede un algoritmo de procesamiento cumplir varias ligaduras de tiempo o debe construirse un hardware especial para hacer el trabajo? ¿Puede un sistema operativo cumplir nuestras necesidades para un manejo eficiente de interrupciones, multitareas y comunicaciones, o debemos construir un ejecutivo a medida? ¿Puede el hardware especificado, acoplado con el software propuesto, cumplir los criterios de rendimiento? Estas y otras muchas preguntas deben ser solucionadas por el ingeniero de sistemas de tiempo real.

## 2. Sistemas en tiempo real

Los sistemas de tiempo real generan alguna acción en respuesta a sucesos externos. Para realizar esta función, ejecutan una adquisición y control de datos a alta velocidad bajo varias ligaduras de tiempo y fiabilidad. Debido a que estas ligaduras son muy rigurosas, los sistemas de tiempo real están frecuentemente dedicados a una única aplicación.

Durante muchos años, los principales consumidores de sistemas de tiempo real eran los militares. Sin embargo, hoy la significativa reducción del coste del hardware ha hecho posible para la mayoría de las compañías, proporcionar sistemas (y productos) de tiempo real para diversas aplicaciones, que incluyen control de procesos, automatización industrial, investigación médica y científica, gráficos de ordenador, comunicaciones locales y de largo alcance, sistemas aeroespaciales, pruebas asistidas por ordenador y un vasto abanico de instrumentación industrial.

### 2.1. Aspectos de integración y de rendimiento

Entre los muchos aspectos relativos al diseño de tiempo real están: la coordinación entre las tareas de tiempo real, el procesamiento de interrupciones del sistema, el manejo de E/S para asegurar que no se pierdan datos, la especificación de las ligaduras de tiempo externas e internas del sistema y el asegurar la precisión de su base de datos.

Cada diseño de tiempo real relativo al software debe ser aplicado en el contexto del “rendimiento” del sistema. En la mayoría de los casos, el rendimiento de un sistema de tiempo real se mide como una o más características relativas al tiempo, pero también se utilizan otras medidas, tales como la tolerancia al fallo.

Algunos sistemas de tiempo real se han diseñado para aplicaciones en las que sólo el tiempo de respuesta o la transferencia de datos es crítica. Otras aplicaciones de tiempo real requieren la optimización de ambos parámetros bajo unas condiciones de cargas extremas. Y, lo que, es más, los sistemas de tiempo real deben manejar sus cargas máximas, mientras se ejecutan varias tareas simultáneamente.

Puesto que el rendimiento de un sistema de tiempo real se determina principalmente por el tiempo de respuesta del sistema y su razón de transferencia de datos, es importante comprender estos dos parámetros. El **tiempo de respuesta** del sistema es el tiempo en el que un sistema debe detectar un suceso interno o externo y responder con una acción. Frecuentemente, la detección del suceso y la generación de la respuesta son tareas sencillas. Es el procesamiento de la información sobre el suceso para determinar la respuesta adecuada lo que puede implicar algoritmos complejos que consumen mucho tiempo.

Entre los parámetros clave que afectan al tiempo de respuesta está el *cambio de contexto* y la *latencia de la interrupción*. El cambio del contexto se refiere al tiempo y sobrecarga necesitado para conmutar entre tareas, y la latencia de interrupción es el tiempo que pasa antes de que el cambio sea realmente posible. Otros parámetros que afectan al tiempo de respuesta son la velocidad de cálculo y el acceso a memorias masivas.

La **razón de transferencia de datos** indica con qué rapidez se introducen o salen del sistema los datos series o paralelos, tanto analógicos como digitales. Los vendedores de hardware, frecuentemente, subrayan los valores de tiempo y capacidad como características de rendimiento. Sin embargo, las especificaciones hardware para el rendimiento son normalmente medidas aisladas y, frecuentemente, dicen poco respecto a la determinación del rendimiento global de un sistema de tiempo real. Por tanto, el rendimiento del dispositivo de E/S, la latencia del bus, el tamaño del buffer, el rendimiento del disco, y una serie de otros factores, aunque importantes, son sólo parte de la historia del diseño de sistemas de tiempo real.

Los sistemas de tiempo real se necesitan normalmente para procesar un flujo continuo de datos de llegada. El diseño debe asegurar que no falte ningún dato. Además, un sistema de tiempo real debe responder a los sucesos que son asíncronos. Por tanto, la secuencia de llegada y el volumen de los datos no pueden predecirse fácilmente de antemano.

Aunque todas las aplicaciones de software deben ser fiables, los sistemas de tiempo real hacen una especial demanda de fiabilidad, reinicialización y recuperación de fallos. Debido a que el mundo real está siendo monitorizado y controlado, la pérdida de monitorización o control (o ambos) es intolerable en muchas circunstancias (p. ej.: un sistema de control de tráfico aéreo). Consecuentemente, los sistemas de tiempo real contienen mecanismos de restauración y recuperación de fallos y, frecuentemente, tienen incorporadas redundancias para asegurar la restauración.

Sin embargo, la necesidad de fiabilidad ha estimulado un continuo debate sobre si los sistemas "interactivos", tales como los sistemas de reservas de billetes y los cajeros automáticos de los bancos, también son de tiempo real. Por otra parte, tales sistemas interactivos deben responder a interrupciones externas dentro de tiempos de respuesta prescritos en el orden de un segundo. Por otra parte, no ocurre ninguna catástrofe si falla uno de esos sistemas en cumplir los requisitos de respuesta; en vez de ello, sólo se consigue una degradación del sistema.

## 2.2. Sistemas operativos de tiempo real (SOTR)

Hoy, dos amplias clases de sistemas operativos se utilizan para los trabajos de tiempo real: **(1)** un SOTR diseñado exclusivamente para aplicaciones de tiempo real y **(2)** sistemas operativos de propósito general que se han reforzado para suministrar capacidades de tiempo real. El uso de un **ejecutivo de tiempo real** hace factible el rendimiento de tiempo real para un sistema operativo de propósito general. Comportándose como software de aplicación, el ejecutivo ejecuta varias funciones del sistema operativo -particularmente las que afectan al rendimiento de tiempo real- de una forma más rápida y eficiente que el sistema operativo de propósito general.

Todos los sistemas operativos deben tener un mecanismo de planificación de prioridades, pero un SOTR debe dar un **mecanismo de prioridades** que permita que las interrupciones de prioridad alta tengan preferencia sobre la menos importante. Además, debido a que las interrupciones ocurren en respuesta a sucesos asíncronos no recurrentes, deben ser servidas sin consumir primero un tiempo de carga de un programa de disco. Consecuentemente, para garantizar el tiempo de respuesta requerido, un sistema operativo de tiempo real debe tener un **mecanismo de bloqueo de memoria**, esto es, mantener unos mínimos programas en memoria principal, de forma que se evite la sobrecarga de almacenamiento en la misma.

Para determinar qué tipo de sistema operativo de tiempo real es más adecuado a una aplicación, pueden definirse y evaluarse medidas de la calidad de SOTR. El tiempo de cambio de contexto y el de latencia de interrupción (discutido anteriormente) determinan la capacidad de manejo de interrupciones, el aspecto más importante de un sistema de tiempo real. El **tiempo de cambio de contexto** es el tiempo que el sistema operativo se toma para almacenar el estado de la computadora y los contenidos de los registros, de forma que pueda volver a la tarea de procesamiento después de servir a la interrupción.

La **latencia de interrupción**, el tiempo máximo que pasa antes de que el sistema pueda conmutar una tarea, ocurre porque en un sistema operativo existen frecuentemente caminos de procesamiento críticos no rentables, que deben ser terminados antes de que pueda procesarse una interrupción. La longitud de estos caminos (el número de instrucciones requeridas antes de que el sistema pueda servir a una interrupción) indica el tiempo perdido en el caso peor. El peor caso se presenta si una interrupción de alta prioridad se genera inmediatamente después de que el sistema entra en un camino crítico entre una interrupción y un servicio de interrupción. Si el tiempo es demasiado largo, el sistema puede perder un trozo irrecuperable de datos. Es importante para el diseñador conocer el tiempo de retraso, de forma que el sistema pueda compensarlo.

Muchos sistemas operativos ejecutan procesamiento multitarea o concurrente, otro requisito importante para los sistemas de tiempo real. Pero para ser viable en la operación en tiempo real, el

solapamiento del sistema debe ser bajo, en términos de tiempo de conmutación y espacio de memoria usado.

### 2.3. Características de los sistemas de tiempo real.

**Realimentación de datos.** Los sistemas de tiempo real suelen tomar acciones sobre el entorno controlado que dependen de una secuencia de datos de entrada-salida y no sólo de las entradas en determinado instante.

**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.

**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.

**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.

**Estabilidad.** Si es imposible cumplir con todas las tareas sin exceder sus restricciones de tiempo, entonces el sistema debe cumplir con las tareas más críticas y de más alta prioridad.

### 2.4. Clasificación de los sistemas de tiempo real.

Según la **especificación para la creación** del software:

- **Sistemas propietarios.** El software se crea para una plataforma determinada dependiendo, por lo general, del sistema operativo y de su arquitectura.
- **Sistemas abiertos.** El software se crea a partir de estándares (de buses, de protocolos de comunicación y de sistemas operativos) que permiten independizar el sistema de la plataforma donde se ejecuta.

Según la **sincronización de los procesos externos** con las acciones realizadas por el ordenador:

- **Sistemas basados en reloj.** La sincronización se define en términos de paso de tiempo.
- **Sistemas basados en sensores.** La sincronización se define en términos de sucesos.
- **Sistemas interactivos.** La relación entre las acciones del computador y la evolución del sistema se define de forma más amplia que en los casos anteriores.

Según los **requerimientos temporales**:

- Sistemas **críticos de tiempo real.** Son sistemas en los que, por su propia naturaleza, se llega a un fallo grave si se incumple alguno de los requisitos temporales.
- **Sistemas acríticos de tiempo real.** Son sistemas en los que es importante el cumplimiento de los requisitos temporales, pero si alguno de ellos no se cumple no se produce fallo.

Según la **arquitectura**:

- **Sistemas centralizados.** Todos los procesadores se encuentran en un único ordenador con lo que se consigue hacer despreciable el tiempo de comunicación entre procesadores frente al tiempo de proceso (multiprocesador de memoria compartida).
- **Sistemas distribuidos.** Los procesadores se encuentran conectados a una red. El tiempo de comunicación entre procesadores no es despreciable frente al tiempo de proceso, por lo que se debe tener en cuenta a la hora de calcular los recursos y sus plazos de respuesta.

Según la composición de los flujos de ejecución:

- **Sistemas monotarea.** Están compuestos por un único flujo de ejecución. El sistema se compone de un bucle infinito en el que se muestran los dispositivos de entrada a una determinada frecuencia y se generan las salidas correspondientes. Su principal ventaja es la sencillez, pero son poco flexibles, lo que impide añadir nueva funcionalidad debido a la alta interdependencia entre las tareas a realizar.
- **Sistemas multitarea.** Están compuestos por un conjunto de tareas (asociadas a procesos que se deben controlar) que se ejecutan de forma concurrente para el control del proceso global. Si es necesario el control de nuevos procesos sólo es preciso añadir nuevas tareas. El principal problema es la planificación de las tareas concurrentes de tal forma que se cumplan los requisitos temporales. En este tipo de sistemas es necesario controlar los recursos y la comunicación entre tareas.

## 2.5. Fidabilidad y tolerancia a fallos

Las exigencias de fiabilidad y la seguridad son mucho más rigurosas en los sistemas de tiempo real que en otros sistemas informáticos. Por ejemplo, si se produce un fallo en una aplicación que calcula la solución de un problema científico, es admisible abortar el programa pues lo único que se habrá perdido es tiempo de cálculo. Sin embargo, en un sistema de tiempo real esto no suele ser una solución aceptable. Un sistema de control por ordenador, por ejemplo, responsable de mantener la temperatura en un horno industrial, no es aceptable detenerlo en cuanto se produce un fallo, pues se podrían alcanzar temperaturas fuera de control que podrían ocasionar la destrucción de la factoría y hasta poner en peligro vidas humanas. Lo mismo ocurre con un sistema que controle un proceso en un reactor nuclear o el sistema de navegación en un avión.

### 2.5.1. Técnicas de programación para la tolerancia a fallos

Todas las técnicas de tolerancia a fallos se basan de alguna forma en la redundancia, es decir, en la introducción de componentes extras en el sistema que permitan la detección de los fallos y su recuperación. Estos componentes son redundantes en el sentido que no son necesarios en un funcionamiento normal. La meta que persiguen las técnicas de tolerancia a fallos es conseguir la máxima fiabilidad con la mínima redundancia.

La redundancia se puede introducir tanto en el hardware como en el software.

## 2.6. Sincronización con sucesos externos

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

Los sistemas de tiempo real tienen que estar operando continuamente mientras estén activos los sucesos sobre los que actúan. Para ello se programan utilizando un bucle infinito del que se sale mediante una condición. La sincronización puede realizarse de diversas maneras:

- **Sondeo.** Se comprueba si ha ocurrido un suceso externo. Si no ha ocurrido, el programa puede continuar con otras acciones antes de volver a comprobar otra vez si se ha producido, o bien puede repetir la comprobación continuamente hasta que ocurre el suceso.
- **Reloj de tiempo real.** Se realizan determinadas acciones a intervalos marcados por el reloj, cuya señal debe comprobarse explícitamente antes de la ejecución de dichas acciones.
- **Interrupciones.** En este caso no se necesita hacer la comprobación de una señal, ya que el ordenador puede estar realizando otras tareas que son suspendidas cuando se produce la interrupción para ejecutar la acción demandada.

## 3. Manejo de interrupciones

Una característica que sirve para distinguir a los sistemas de tiempo real de cualquier otro tipo es el manejo de interrupciones. Un sistema de tiempo real debe responder a un estímulo externo - interrupción- en un tiempo dictado por el mundo externo. Debido a que, frecuentemente, se presentan múltiples estímulos (interrupciones), deben establecerse prioridades e interrupciones prioritarias. En otras palabras, la tarea más importante debe siempre ser servida dentro de las ligaduras de tiempo predefinidas, independientemente de otros sucesos.

El manejo de interrupciones supone, no sólo almacenar información, de forma que el ordenador pueda restablecer correctamente la tarea interrumpida, sino también evitar interbloqueos y bucles sin fin.

### 3.1. Concepto. Interrupciones múltiples.

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.

### 3.2. 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.

El flujo normal de procesamiento es "interrumpido" por un suceso que es detectado por el hardware del procesador. Un suceso es cualquier ocurrencia que necesita un servicio inmediato y puede ser generado por hardware o por software. Se salva el estado del programa interrumpido (es decir, se guardan todos los contenidos de los registros, los bloques de control, etc.) y se pasa el control a una rutina de servicio de interrupción, que bifurca al software apropiado para el manejo de la interrupción. Al terminar el servicio de la interrupción, se restaura el estado de la máquina y continúa el flujo normal de procesamiento.

En muchas situaciones, el servicio de interrupción de un suceso puede a su vez ser interrumpido por otro suceso de mayor prioridad. Pueden establecerse niveles de prioridad de interrupciones. Si se permite accidentalmente a un proceso de prioridad más baja interrumpir a uno de prioridad mayor, puede ser difícil restaurar los procesos en el orden correcto y puede producirse un bucle sin fin.

Para manejar las interrupciones y cumplir también las ligaduras de tiempo del sistema, muchos sistemas operativos de tiempo real hacen cálculos dinámicos para determinar si pueden cumplirse los objetivos del sistema. Estos cálculos dinámicos se basan en la frecuencia media de ocurrencia de sucesos, la cantidad de tiempo que le lleva el servirlos (si pueden ser servidos) y las rutinas que pueden interrumpirlos y temporalmente evitar su servicio.

Si los cálculos dinámicos muestran que es posible manejar los sucesos que pueden ocurrir en el sistema y también cumplir las ligaduras de tiempo, el ingeniero de sistemas debe decidir sobre un esquema de acción. Un posible esquema consiste en almacenar en un buffer los datos, de forma que puedan ser procesados rápidamente cuando el sistema esté preparado.

## 4. Sincronización y comunicación de tareas

Un sistema de multitarea debe suministrar un mecanismo por el que las tareas se pasen información unas a otras, así como para asegurar su sincronización. Para estas funciones, los sistemas operativos y los lenguajes con soporte de tiempo real, utilizan frecuentemente semáforos de colas, buzones o sistemas de mensajes.

El problema más general que debe resolverse en el tratamiento de los procesos concurrentes, es el de la comunicación, que consiste en proporcionarles mecanismos que les permitan intercambiar información.

### 4.1. Semáforos

Los **semáforos** suministran sincronización y señalización, pero no contienen información. Los **mensajes** son similares a los semáforos, excepto en que ellos llevan una información asociada. Por otra parte, los **buzones** no señalizan la información, sino que la contienen.

Los **semáforos de colas** son primitivos de software que ayudan a gestionar el tráfico. Suministran un método para dirigir varias colas -por ejemplo, colas de tareas en espera de recursos, acceso a base de datos o dispositivos, así como colas de recursos y dispositivos. Los semáforos coordinan

(sincronizan) las tareas en espera con lo que estén esperando, sin dejar que las tareas o recursos interfieran entre sí.

#### 4.2. Memoria Compartida

Los procesos comparten algunas variables para el intercambio de información. La responsabilidad de proporcionar la comunicación recae sobre los programadores de aplicaciones, el sistema operativo únicamente tiene que ofrecer la memoria compartida.

#### 4.3. Intercambio de mensajes

Un tercer método para la comunicación y sincronización entre procesos es un sistema de mensajes. Con un sistema de mensajes, un proceso envía un mensaje a otro. El último se activa entonces automáticamente por el sistema de soporte de tiempo de ejecución o sistema operativo para que procese el mensaje. Tal sistema incurre en sobrecarga debido a la transferencia real de la información, pero suministra una mayor flexibilidad y facilidad de uso.

Se utilizan dos primitivas:

```
enviar (destinatario, mensaje);  
recibir (origen, mensaje);
```

Se trata de llamadas al sistema y no comandos del lenguaje por lo que son accesibles desde muchos entornos de lenguajes de programación.

Un envío con bloqueo debe esperar hasta que el receptor reciba el mensaje constituyendo un ejemplo de comunicación síncrona. Un envío sin bloqueo permite al transmisor continuar con otras tareas, aunque el receptor no haya recibido aún el mensaje permitiendo pues que ambos interlocutores se comuniquen asíncronamente. El envío sin bloqueo requiere de buffers para almacenar los mensajes hasta que los reciba el receptor. La comunicación asíncrona aumenta la posibilidad de paralelismo.

En la transmisión de mensajes pueden darse errores en la comunicación y es necesario un protocolo que permita solucionar este problema. También es necesario poder identificar mediante un nombre de forma no ambigua a los procesos destinos y origen. Estos y otros aspectos relativos a la comunicación son tratados en otros temas del temario por lo que no nos extendemos más sobre ellos.

#### 4.4. Buzones y puertos

Un buzón consiste en un buffer en el que se almacenan los mensajes enviados antes de ser extraídos por el receptor. En este caso, el transmisor y el receptor especifican el buzón, en lugar del proceso, con el que se van a comunicar.

Un puerto se define como un buzón utilizado por múltiples transmisores y un único receptor. En los sistemas cliente/servidor, cada servidor tiene un puerto asignado por el cual recibe los mensajes de multitud de clientes.

En un sistema de tiempo real, los semáforos se utilizan frecuentemente para implementar y gestionar buzones. Los buzones se almacenan temporalmente en lugares (también llamados buffers o almacenes de mensajes) para enviar mensajes de un proceso a otro. Un proceso produce una información, la sitúa en el buzón y luego señala a un proceso consumidor que hay una información en el buzón para que la utilice.

Algunos métodos para los sistemas operativos de tiempo real o sistemas de soporte en tiempo de ejecución ven los buzones como la forma más eficiente de implementar comunicaciones entre procesos. Algunos sistemas de tiempo real suministran un lugar para enviar y recibir referencias a los datos del buzón. Esto elimina la necesidad de transferir todos los datos -ahorriendo así tiempo y sobrecarga.

#### 4.5. Conductos (pipes)

UNIX introdujo la noción de conductos como un esquema para manejar comunicaciones entre procesos. Un conducto es en esencia un buzón que permite extraer un número específico de bytes a la vez. El conducto no mantiene las fronteras entre mensajes. Si los procesos convienen en leer y escribir mensajes con un tamaño determinado o terminar cada mensaje con un señalador especial la utilización de los conductos no presenta ningún problema.

#### 4.6. Llamadas a procedimientos remotos (RPC)

Son un mecanismo estructurado de alto nivel para realizar la comunicación entre procesos en sistemas distribuidos. Con una llamada a un procedimiento remoto, un proceso de un sistema A llama a un procedimiento de un proceso de otro sistema B. El proceso A se bloquea esperando el retorno desde el procedimiento llamado en el sistema remoto y después continúa su ejecución desde el punto que sigue inmediatamente a la llamada.

El procedimiento llamado y el que llama residen en procesos distintos, con espacios de direcciones distintos, por lo cual no existe la noción de variables globales compartidas, por lo que las RPC transfieren la información estrictamente a través de los parámetros de la llamada.

Un problema de las RPC es que las llamadas por referencia son difíciles de implementar ya que una RPC comunica espacios de direcciones diferentes.

#### 4.7. Bases de datos de tiempo real

Como muchos sistemas de procesamiento de datos, los sistemas de tiempo real, frecuentemente, van junto con una función de gestión de base de datos. Sin embargo, puede parecer que las bases de datos distribuidas constituyen el método preferido en los sistemas de tiempo real, debido a que la multitarea es muy común y que los datos se procesan frecuentemente en paralelo. Si la base de datos es distribuida y no centralizada, las tareas individuales pueden acceder a sus datos de forma más rápida, fiable y con menos cuellos de botella. El uso de una base de datos distribuida para

aplicaciones de tiempo real divide el tráfico de entrada/salida y acorta las colas de las tareas en espera, para acceder a una base de datos. Además, un fallo de una base de datos raramente causará el fallo del sistema entero si se construyen con redundancia.

## 5. Lenguajes de tiempo real

Debido a los requisitos especiales de rendimiento y de fiabilidad demandados por los sistemas de tiempo real, la elección del lenguaje de programación. Pueden usarse con efectividad muchos lenguajes de programación de propósito general (p. ej.: C, FORTRAN, Modula-2) para aplicaciones de tiempo real. Sin embargo, una clase llamada lenguajes de tiempo real (p. ej.: Ada, Jovial, HAL/S, Chill y otros) se utilizan frecuentemente en aplicaciones especializadas de comunicaciones y militares.

Varias características hacen un lenguaje de tiempo real diferente de un lenguaje de propósito general. Estas incluyen la capacidad de multitarea, construcciones para implementación directa de funciones de tiempo real y características modernas de programación que ayuden a asegurar la corrección del programa.

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 SOTR dan capacidad de multitarea, frecuentemente existe software de tiempo real empotrado sin un sistema operativo. En vez de ello, las aplicaciones empotradas se escriben en un lenguaje que da un soporte 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 una multitud de otras construcciones de control y definición de datos.

### 5.1. Lenguajes utilizados para la programación en tiempo Real

#### 5.1.1. ADA

El lenguaje Ada se caracteriza por hacer énfasis en la fiabilidad. Es el lenguaje mayoritario en sistemas de seguridad crítica como gestión aeronáutica, tráfico aéreo y ferroviario, etc.

Es uno de los pocos lenguajes que tiene soporte directo para programar sistemas de tiempo real. ADA también soporta la programación orientada a objetos, la programación concurrente etc...

### 5.1.2. JAVA

De igual forma Java es un lenguaje moderno que soporta la programación orientada a objetos, la programación concurrente y también tiene soporte opcional para sistemas de tiempo real mediante la librería RTSJ: Real-Time Specification for Java.

Java soporta programación multihilo con características de tiempo real, planificación por prioridades, aunque tiene una dificultad importante en la gestión de la memoria: La memoria automática de Java no es apta para sistemas de tiempo real; RTSJ define otras formas de gestión de memoria. Existe mucho interés en RTSJ, pero escasas aplicaciones, aún.

### 5.1.3. C

Desarrollado en 1972 a partir de otro lenguaje llamado B, con la idea de conseguir un lenguaje de alto nivel, pero con posibilidad de acceso a bajo nivel. Esta propiedad hace que este lenguaje sea muy adecuado para la programación de sistemas.

Es adecuado para programadores expertos porque hace programas muy eficientes, por el contrario, su depuración puede ser bastante compleja. Tiene las siguientes características:

- Es débilmente tipado
- Impone pocas restricciones al programador
- Dispone de operadores a nivel de bit
- Soporta el uso de punteros
- Utiliza librerías

C es utilizado ampliamente en sistemas embebidos y en tiempo real. Generalmente se utilizan bibliotecas de funciones específicas para este tipo de programación.

## 6. Conclusión

Aplicaciones específicas de alto rendimiento, y de respuesta crítica y precisa, requieren una programación específica con el uso de interrupciones, sincronización y comunicación entre tareas con asignación de prioridades en tiempo real.

Actualmente lenguajes de programación de propósito general como C han perdido el encanto de la programación a alto nivel, para desarrollarse a más bajo nivel en programación de sistemas embebidos y en tiempo real, con librerías que permitan el manejo de funciones y tareas básicas del sistema operativo.

## 6.1. Relación del tema con el sistema educativo actual

Este tema puede ser desarrollado en los siguientes módulos formativos (para atribución docente de PES)

- Bachillerato – Tecnologías de la Información y la Comunicación II (PES)
- GS- GS – DAW – Desarrollo Web en Entorno Servidor DAW/DAM – Programación (PES)

Aunque los profesores de SAI no tienen una atribución docente en ningún módulo con aplicación directa del tema, siempre es probable que se diseñe algún pequeño programa en clase, para lo que recurrir a la notación de pseudocódigo o de ordinograma sería un recurso esclarecedor.

## 7. Bibliografía

- Introduction to Java Programming and Data Structures, Comprehensive Version. Y. Daniel Liang. Pearson, 11<sup>a</sup> edición. 2017.
- Java 9. Francisco Javier Moldes Teo. Anaya. 2017.
- Introducción a la computación. J. Glenn Brookshear, Pearson, 11<sup>º</sup> edición. 2012
- Fundamentos de programación. Algoritmos, estructuras de datos y objetos. Luis Joyanes Aguilar, McGraw-Hill, 4<sup>º</sup> edición. 2008.
- Langsam, Augenstein y Tanembaum: “Estructuras de Datos con C y C++”, Prentice-Hall 1997
- Prieto A., Lloris A. y Torres J.C.: Introducción a la Informática, 4<sup>a</sup> ed (2006) McGraw-Hill
- Lenguajes de programación, principios y práctica. 2<sup>a</sup> Ed (2004). Kenneth C.Louden

