Sistemas y Aplicaciones Informáticas

Tema 30. Programación en Tiempo Real. Interrupciones. Sincronización y Comunicación entre Tareas.

1. Ámbito de docencia.

- Sistemas informáticos monousuario y multiusuario (ASI 1).
- Sistemas informáticos multiusuario y en red (DAI 1).
- Sistemas operativos en entornos monousuario y multiusuario (ESI 1).

2. Programación en tiempo real.

2.1. Sistemas de tiempo real.

2.1.1. Concepto. Parámetros de rendimiento.

- 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.

2.1.2. 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.

2.1.3. Clasificación de los sistemas de tiempo real.

- Según la <u>especificación para la creación del software</u>:
  - *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 <u>sincronización de los procesos externos con las acciones realizadas por el ordenador</u>:
  - *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.

2.2. Programación de sistemas de tiempo real.

2.2.1. Concepto. Diferencia con otros tipos de programación.

- 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.
- Se diferencia de los otros tipos de programación en lo siguiente:
    - *Programación secuencial*. Es el tipo de programación clásica, en ella las acciones a realizar se ordenan de forma consecutiva. El comportamiento del programa depende sólo de los efectos de las acciones individuales y de su orden, no teniendo ninguna importancia el tiempo que lleve realizar una determinada acción.
    - *Programación multitarea*. Es un tipo de programación por la cual los programas pueden estar formados por varios procesos que pueden ser parcialmente secuenciales, pero que se ejecutan de forma concurrente. Los procesos se comunican entre sí utilizando variables compartidas y señales de sincronización.

2.2.2. Características específicas.

2.2.2.1. Implementación de la concurrencia.

- **Ejecutivo cíclico**. Un programa consta de un solo bucle que se repite con una frecuencia determinada, guiado por un reloj. En cada pasada se ejecuta el proceso correspondiente a las tareas que tengan dicha frecuencia. Cada dos pasadas, las tareas de frecuencia la mitad, y así sucesivamente. Requiere que todas las tareas sean periódicas de frecuencias armónicas, y es difícil de implementar y modificar.
- **Guiada por interrupciones**. El programa consiste en una serie de manejadores de interrupción lanzados por eventos externos asíncronos. Se trata de pseudoconcurrencia ya que no existen mecanismos de comunicación y sincronización entre tareas.
- **Soportada por el lenguaje de programación**. El programa soporta la gestión de múltiples tareas, con concurrencia real en el caso de sistemas multiprocesadores. Requiere la utilización de mecanismos de comunicación y sincronización entre tareas.

2.2.2.2. Control de las excepciones.

- Un programa de tiempo real debe reaccionar adecuadamente ante eventos imprevistos. Una excepción es una manifestación de un error. Existen varias formas de detectar excepciones:
  - Por el valor de retorno de las rutinas.
  - Por modificación de la dirección de retorno por la propia rutina.
  - Por salto global a un punto del programa cuando se produce una excepción.
  - Por mecanismos provistos por el lenguaje de programación.
- Cuando se produce una excepción en un determinado bloque de código, se interrumpe la ejecución del programa y se transfiere el control al manejador de excepciones asociado a dicho bloque. El manejador de excepciones es un código que realiza el tratamiento de una o varias excepciones. Para poder establecer una conexión entre las excepciones y los manejadores, a cada clase de excepción se le da un nombre. Ciertas excepciones pueden estar predefinidas en el lenguaje y otras pueden ser definidas por el programador.
- Si no existe un manejador asociado al bloque donde se produjo la excepción, entonces ésta se propaga a lo largo de la cadena de bloques llamadores sucesivamente hasta encontrar un manejador adecuado. Si la propagación lleva la excepción hasta el programa principal y no hay manejador, el programa es abortado. Una vez tratada la excepción hay dos opciones:
  - *Terminación*. El bloque que produjo la excepción no se vuelve a ejecutar. El control pasa a quien llamó a ese bloque, es el modelo más común.
  - *Reanudación*. El bloque que produjo la excepción vuelve a ejecutarse a partir de la sentencia que provocó la excepción o a partir de la siguiente.

2.2.2.3. Mecanismos de fiabilidad y 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. Interrupciones.

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.

4. Sincronización y comunicación entre tareas.

4.1. Conceptos previos.

4.1.1. Sección crítica. Exclusión mutua.

- 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.

4.1.2. Conceptos de sincronización y comunicación entre tareas.

- 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.

4.2. Mecanismos de sincronización y comunicación.

4.2.1. 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.

4.2.2. 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 (S: Semaforo; v: integer).* Poner el valor del semáforo S al valor de v. Se debe llevar a cabo antes de que comience la ejecución concurrente de los procesos ya que su función exclusiva es dar un valor inicial al semáforo.
  - *espera (S).* Si S > 0 entonces S := S - 1 y continuar, si no dejar en espera la tarea que hace la llamada y ponerla en la cola de tareas. Un proceso que ejecuta la operación *espera* y encuentra el semáforo con un valor positivo, lo decrementa y prosigue su ejecución. Si el semáforo está a cero, el proceso queda en estado de espera hasta que el semáforo se libera.
  - *señal (S).* Si la cola de tareas está vacía entonces S := S + 1, si no reanudar la primera tarea de la cola de tareas. Cuando un proceso ejecuta la operación *señal*, el proceso que abandona la cola de tareas para pasar al estado listo dependerá del esquema de gestión de la cola de tareas suspendidas que se haya implementado en el diseño del semáforo. Si no hay ningún proceso en espera del semáforo, éste se deja libre (S > 0) para el primero que lo requiera.
- 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.

4.2.3. 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, lo que 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.

4.2.4. 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.

El buzón establece un enlace que puede ser utilizado por más de dos procesos y permite que la comunicación de un proceso con otro se pueda realizar mediante distintos buzones. Si hay varios procesos que recogen información del mismo buzón se pueden dar distintas soluciones: permitir que un buzón sólo pueda ser compartido por dos procesos, permitir que sólo un proceso cada vez pueda ejecutar una operación de recibir y, por último, que el sistema identifique al receptor del mensaje.

- 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. El proceso emisor se suspende hasta que la ejecución de una operación de recibir por parte del receptor le saca de ese estado, procediendo al envío del mensaje. Desde el momento en que el emisor puede seguir adelante, sabe que su mensaje ha sido recibido. De este modo una pareja emisor-receptor no puede tener más de un mensaje pendiente en cada momento.
  - *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. La implementación de la cola se realiza normalmente con una capacidad de mensajes finita mayor que cero.
  - *Invocación remota*. El proceso que envía el mensaje sólo prosigue su ejecución cuando ha recibido una respuesta (recibido o error) del receptor. Se especifica en la sentencia de recibir un tiempo máximo de espera del mensaje. Si transcurre el tiempo especificado el sistema operativo desbloquea al proceso emisor suspendido, y le envía un mensaje o código de error indicando el agotamiento del tiempo de espera. La invocación remota se puede construir a partir de dos comunicaciones síncronas.
- 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.