Sistemas en Tiempo Real

Grado en Ingeniería Electrónica

UNIDAD 9: Capacidades de Tiempo Real

Capítulo 9 de la edición Inglés (2009)

Capítulo 12, excepto punto 12.8 de la edición en Español (2003)

Tutoría

1. La noción del tiempo
2. Acceso al reloj
3. Retardo de tareas
4. Programación de timeouts
5. Ámbitos temporales

Noción del tiempo

Los requisitos temporales constituyen la característica diferenciadora de los sistemas de tiempo real, ya que necesitan coordinar su ejecución con el tiempo de su entorno.

La **noción de tiempo** se relaciona con los siguientes aspectos:

- Acceso a un reloj para medir el paso del tiempo.
- Retardo, retraso o desfase de las tareas.
- Tiempos límite de espera (timeouts).
- Especificación y planificación de estos tiempos límite.

Es necesario medir el paso del tiempo, poder parar una tarea, sin que tenga ocupada la CPU, y cumplir unas restricciones temporales.

Para facilitar la especificación de dichas restricciones y requisitos temporales, se utiliza el concepto de **ámbito temporal.**

Acceso a un reloj

Un programa que debe interactuar con el marco temporal debe poder acceder a métodos que devuelvan la hora y que permitan medir el paso del tiempo. Esto se puede hacer de dos formas:

1. Accediendo directamente al marco temporal del entorno:
    - El entorno puede proporcionar una interrupción regular sincronizada internamente con el reloj.
    - El sistema puede sintonizar alguna señal externa de tiempo internacional.

2. Mediante un reloj hardware interno que ofrezca una aproximación adecuada del paso del tiempo.

    Estos relojes son dispositivos que cuentan el número de oscilaciones de un cristal de cuarzo, pero no siempre sincronizan exactamente con referencias externas del tiempo.

En los **sistemas distribuidos** hay que tener en cuenta los desvíos con el tiempo externo, y también las diferencias entre todos los relojes internos del sistema.

Si cada nodo tiene su propio reloj, no existe una fuente perfecta del tiempo.

Si se utiliza un servicio de tiempo global, también será necesario un protocolo de sincronización de relojes.

Los paquetes de reloj en Ada

Paquete **Calendar** (obligatorio):

- Implementa un **TAD Time**
  - Función **Clock** para leer el tiempo
  - Conversiones de Time a unidades de tiempo comprensibles:
    - Year, Month, Day (enteros), Seconds (subtipo del tipo **Duration**, real de punto fijo que representa un intervalo de tiempo relativo interpretado como una cantidad de segundos)

Paquete Real_Time (opcional):

- Proporciona mayor precisión
  - **Time_Unit** es la cantidad más pequeña de tiempo representada por el **tipo Time**: su rango debe ser desde el inicio del programa hasta al menos 50 años.
  - **Tick** (pulso) no debe ser mayor de un milisegundo.

Los paquetes de reloj en Java RealTime

Paquete **java.lang**:

System.currentTimeMillis(): método estático que devuelve **la hora actual del sistema** en milisegundos (desde las 0:00 del 1/1/1970)

Paquete javax.realtime:

- Abstract class **HighResolutionTime**: métodos para leer, escribir y comparar valores de tiempo con precision de nanosegundos.

  Tiene 2 subclases**:**

  - **AbsolutTime**
  - **RelativeTime** (similar al **Duration** de Ada)

  También cuenta con el método **absolute()**: para convertir el tiempo que se le pasa a tiempo absoluto relativo a algún reloj

- Abstract class **Clock** de la que derivan todos los relojes

Relojes en C/Real-Time POSIX

ANSI C tiene una librería estándar con varias funciones para obtener la hora actual del calendario, con diferentes niveles de resolución.

Define un tipo de tiempo básico **time_t** y varias rutinas para la manipulación de objetos de este tipo.

POSIX cuenta con una función **clock_gettime,** que puede recuperar la hora con una precisión de hasta nanosegundos, desde una variedad de relojes diferentes. Los relojes pueden ser de todo el sistema, midiendo el mismo tiempo para todas las tareas; o pueden ser por tarea.

El valor devuelto por **clock_gettime** está definido por la estructura **timespec**, donde **tv-sec** y **tv-nsec** indican los segundos y nanosegundos transcurridos desde el 1/1/1970.

Todos los relojes son de tipo **clockid_t. POSIX** permite que una implementación soporte varios relojes, y es necesario definir una constante de tipo **clockid_t CLOCK_REALTIME**.

Retraso o retardo de un proceso o tarea

Además de poder acceder a un reloj, las tareas también deben ser capaces de retrasar su ejecución, ya sea por un periodo fijo de tiempo o hasta cierto instante absoluto de tiempo futuro.

Retardo relativo

Posibilita que una tarea sea encolada hasta cierto evento futuro en lugar de efectuar una espera ocupada basada en llamadas al reloj.

En **Ada**: sentencia **delay**

En **C/RealTime POSIX**: llamada del sistema **sleep o nanosleep** (dependiendo de la precisión requerida)

En **Java**: El método **sleep** de la clase **Thread** permite que una tarea se retrase determinados milisegundos.

La clase **RealtimeThread** permite precisión de alta resolución con respecto a un reloj.

Retardo absoluto

En lugar de especificar el retardo desde el momento actual, utiliza alguna referencia.

En **Ada**: **delay until**

En **POSIX**: se pueden construir retardos absolutos utilizando un **temporizador** y esperando a la señal que se genera cuando expira.

En **Java**: **sleep** de **RealTimeThread** sirve tanto para retardos relativos como absolutos

Programación de timeouts (tiempos límite de espera)

La restricción temporal más simple que puede tener un sistema embebido es reconocer y actuar sobre la **no ocurrencia** de cierto suceso externo.

El **tiempo límite de espera (timeout)** es una restricción sobre el tiempo que una tarea está dispuesta a esperar por una **comunicación, y también son necesarios** en la **ejecución de acciones**.

Variables compartidas y timeouts

Cuando una tarea intenta acceder a una **sección crítica**, se bloquea si hay otra tarea activa en la sección. Este bloqueo está limitado por el tiempo necesario para ejecutar el código de la sección y el número de procesos que quieren ejecutar dicha sección, por lo **que no se considera necesario asociar un timeout al intento de entrada a sección crítica**.

Sin embargo, es muy importante establecer **timeouts** en una **sincronización condicional**, ya que la tarea por la que se espera puede retrasarse un periodo de tiempo importante.

Paso de mensajes y timeouts

Con el paso de mensajes síncrono, cuando una tarea se involucra en una comunicación debe esperar por la respuesta. Esto implica establecimiento de **timeouts**.

Programación de timeouts (tiempos límite de espera)

Timeouts en acciones

Un **timeout** puede verse como un tipo de notificación, por lo que, si se soportan notificaciones asíncronas, se pueden utilizar **timeouts**, tanto en el modelo de reanudación (asíncrono) como en el de terminación (asíncrono).

Para usar **timeouts** en acciones se necesita el modelo de terminación.

Por ejemplo, una tarea puede ser abortada si sucede un evento antes de que haya finalizado… el paso del tiempo puede ser ese evento.

Además de establecer timeouts, hay que tener en cuenta consideraciones importantes sobre el tiempo.

Para la verificación de un sistema en Tiempo Real sería necesario responder a dos aspectos:

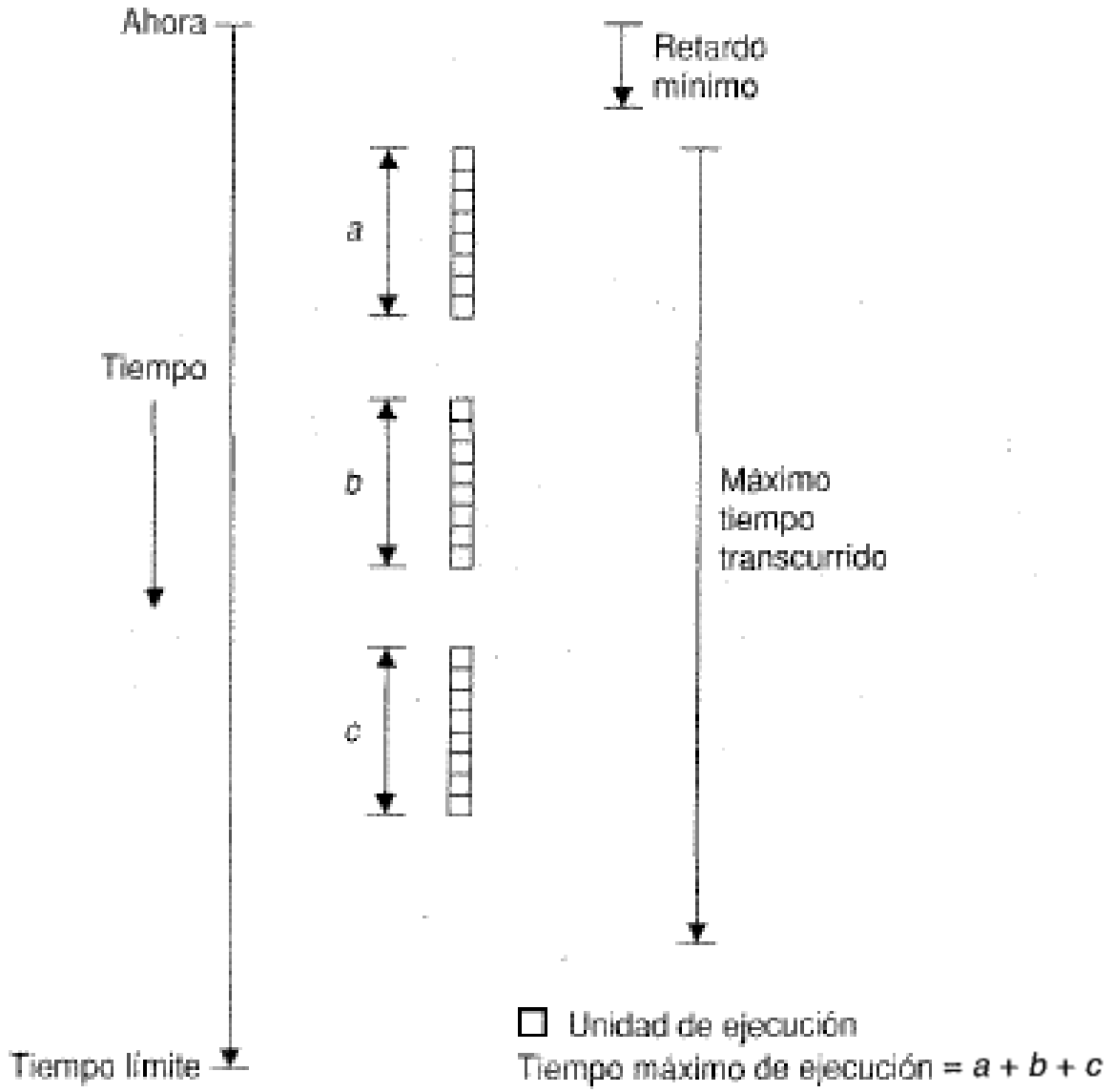
- Fase de diseño: ¿Los requisitos temporales son coherentes y consistentes y se pueden satisfacer?
- Fase de implementación: ¿Se pueden satisfacer esos requisitos temporales con el hardware disponible?

Ámbitos temporales

Se utilizan para facilitar la especificación de las restricciones temporales.

Los atributos de un AT incluyen:

- **Límite temporal o tiempo límite (deadline):** el instante de tiempo en el cual la ejecución de un AT debe haber finalizado.
- **Retardo mínimo:** la mínima cantidad de tiempo que debe pasar antes del comienzo de la ejecución de un AT.
- **Tiempo máximo de ejecución** de un AT.
- **Lapso de tiempo máximo (tiempo máximo transcurrido)** desde que comience la ejecución de un AT.



Ámbitos temporales

Los ámbitos temporales pueden ser:

Periódicos

Toman datos o ejecutan un bucle de control, y tienen un **tiempo límite (deadline)** que deben cumplir.

Esporádicos

Suelen ser consecuencia de eventos asíncronos externos y llevan asociados **tiempos de respuesta específicos**.

En muchos lenguajes de tiempo real, los **ámbitos temporales** están asociados a las **tareas** que los incorporan, de manera que las tareas se pueden describir como **periódicas**, **aperiódicas** o **esporádicas**, dependiendo de las propiedades de sus ámbitos temporales internos.

Principalmente hay que conseguir:

- Ejecución de tareas periódicas a un ritmo correcto.
- Ejecución de todas las tareas dentro de sus límites temporales.

Esto se traduce en que es necesario planificar las tareas teniendo en cuenta los tiempos límite (deadlines).

Ámbitos temporales

Un sistema de tiempo real se dice que es **estricto (hard)** si tiene tiempos límite (deadlines) cuyo incumplimiento producirá el fallo del sistema.

Por el contrario, un sistema de tiempo real es **no estricto (soft)** si la aplicación es tolerante con el incumplimiento de los tiempos límite (deadlines).

Un sistema es simplemente **interactivo** si no tiene fijado ningún tiempo límite (deadline) pero se esfuerza por mantener tiempos de respuesta adecuados.

Una tarea con un tiempo límite no estricto (soft deadline) puede prestar su servicio con retraso, es decir, el sistema seguirá obteniendo algún valor del servicio tardío, pero si una tarea **no estricta** tiene establecido algún **tiempo límite rígido** se dice que es **firme**.

En el caso de los **sistemas tolerantes a fallos**:

- **Estricto (hard):** el incumplimiento de un tiempo límite (deadline) dispara una rutina de recuperación de errores.
- **No estricto o firme:**
  - Ocasionalmente se puede incumplir algún tiempo límite (deadline).
  - O, se admiten incumplimientos no demasiado grandes de tiempos límite (deadlines).

MODULO 3: TEMPORIZACIÓN DE LAS TAREAS DE UN SISTEMA EN TIEMPO REAL

Este módulo explica los mecanismos de temporización que se pueden aplicar en un sistema de tiempo real, tomando como base el reloj del sistema en tiempo real. Una vez definidos los conceptos necesarios (timeouts, ámbitos temporales, etc.) se explica la forma de categorizar las tareas para su inclusión y priorización en la programación temporal del sistema. Una vez hecho esto, se pueden aplicar varias estrategias de planificación basadas en prioridades o tiempos de ejecución. Se termina el módulo mostrando los detalles específicos de programación sobre entornos de ejecución asociados a los mecanismos de entrada/salida de dichos entornos.

Foro de dudas del módulo 3 — Marcar como hecha

Dudas PED3

Dudas PED4

Autoevaluaciones de las unidades

Autoevaluación de la Unidad 9: Capacidades de tiempo real

Autoevaluación de la Unidad 10: Planificación de sistemas de tiempo real

Autoevaluación de la Unidad 11: Programación de bajo nivel

Ejercicios resueltos sobre el planificador de ciclo ejecutivo

ejercicio1.pdf

ejercicio2.pdf

Prácticas

PED3: Desarrollo de código RT

Prueba de evaluación a distancia (PED3): Desarrollo de código RT

En el fichero zip, están tanto el enunciado (PED3.pdf) como el ejemplo de implementación del PID (PID.pdf)

PED4: Real time sensor y AWS IoT (integración de código)

Prueba de evaluación a distancia (PED4): Integración del código RT

Leer el enunciado adjunto

Sistemas en Tiempo Real

Grado en Ingeniería Electrónica

UNIDAD 9: Capacidades de Tiempo Real

UNED

Nieves Pedreira Souto