SISTEMAS DE TIEMPO REAL (RTS)

Sistemas de Tiempo Real:

Es aquel que el ***resultado correcto*** (ante un evento tener siempre la misma respuesta) depende tanto de su ***validez lógica*** como así también ***del instante*** en que se produce (determinismo).

*Determinismo: El tiempo de ejecución de los programas de tiempo real deben estar **acotado** en el caso más desfavorable para cumplir las restricciones temporales. (No debe ser necesariamente rápido).*

En estos tipos de sistemas es muy frecuente encontrar un *software* dividido en varios niveles jerárquicos, estos son 4:

- ***Adquisición de datos y actuadores***. Este es el nivel más bajo en la jerarquía, encargado de interactuar con el hardware del sistema a controlar. El software de este nivel suele ser corto y normalmente es necesario ejecutarlo frecuentemente. La ejecución suele estar controlada por un temporizador o por una señal externa generada por el propio hardware. En ambos casos se usan interrupciones para detectar el fin del temporizador o la activación de la señal externa

- ***Algoritmos de control (PID).*** Este nivel se corresponde con los algoritmos de control que, a partir de las medidas obtenidas por el sistema de adquisición de datos, ejecutan un algoritmo de control digital; como por ejemplo un control PID o un control adaptativo más sofisticado. El resultado de sus cálculos es el valor que hay que enviar a los actuadores de la planta. Normalmente este *software* ha de ejecutarse con un periodo de tiempo determinado (periodo de muestreo).

- ***Algoritmos de supervisión (trayectorias).*** Existen sistemas de control sofisticados en los que existe una capa de control a un nivel superior que se encarga de generar las consignas a los controles inferiores. Por ejemplo, pensemos en un controlador para un brazo robot. Existe un nivel superior que se encarga de calcular la trayectoria a seguir por el brazo y en función de ésta, genera las consignas de velocidad para los motores de cada articulación. Al igual que los algoritmos de control, estos algoritmos se ejecutan también periódicamente, aunque normalmente con un periodo de muestreo mayor que los algoritmos de control de bajo nivel.

- ***Interfaz de usuario, registro de datos, etc.*** Además del software de control, es normalmente necesario ejecutar otra serie de tareas como por ejemplo un interfaz de usuario para que éste pueda interactuar con el sistema; un registro de datos para poder analizar el comportamiento del sistema en caso de fallo, comunicaciones con otros sistemas, etc. Al contrario que en resto de niveles, en este no existen restricciones temporales estrictas, por lo que se ejecuta cuando los niveles inferiores no tienen nada que hacer.

Entonces normalmente yo lo puedo implementar como un programa secuencial que se divide en funciones que se ejecutan según un orden preestablecido.

Un sistema en tiempo real se divide en tareas que se ejecutan en “paralelo”.

La ejecución en paralelo (o pseudo paralelismo) se consigue como la sucesión rápida de actividades secuencial (multitasking – multitarea).

Métodos para implantar un sistema en tiempo real:

- Procesamiento secuencial (Bucle de scan).
- Primer plano / segundo plano (Foreground/Background).
- Multitarea cooperativa.
- Multitarea expropiativa (preemptive).

Procesamiento Secuencial

La técnica de procesamiento secuencial consiste en ejecutar todas las tareas consecutivamente una y otra vez dentro de un bucle infinito. A esta técnica se le denomina también bucle de *scan* debido a que está basada en un bucle infinito que ejecuta una y otra vez todas las tareas.

La característica fundamental de este tipo de sistemas es que las tareas ***no pueden bloquearse*** a la espera de un evento externo. Ello es debido a que estos eventos externos son asíncronos con el funcionamiento del sistema, por lo que el tiempo que va a tardar la tarea que espera el evento no está acotado. En consecuencia, no puede garantizarse el tiempo que va a tardar el bucle de *scan* completo en terminar un ciclo, violándose entonces el principio fundamental de un sistema en tiempo real, que dice que el tiempo de respuesta máximo ha de estar garantizado.

Temporización del bucle de scan

Es fácil obtener un tiempo de muestreo exacto siempre y cuando el periodo sea ***mayor que la suma de los tiempos máximos de ejecución de las tareas***. Para ello, basta con añadir un temporizador (timer) al hardware del sistema y añadir una última tarea que espera el final del temporizador. Esta tarea se suele denominar tarea inactiva (idle). La tarea constará simplemente de un bucle de espera que estará continuamente comprobando si ha finalizado la cuenta del temporizador.

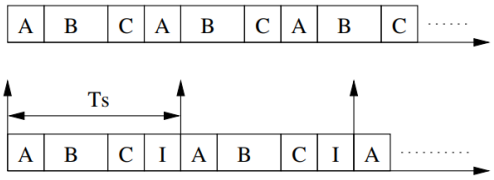


Figura 1.3: Temporización de tareas.

Una vez finalizada la cuenta, la tarea reinicia el temporizador y le devuelve el control al programa principal, con lo que volverá a empezar el ciclo de scan. Como puede observar en la figura, se ha añadido una tarea inactiva (I) que se queda a la espera del final del temporizador. De esta forma se consigue ***un periodo de muestreo independiente del tiempo de ejecución de las tareas y del hardware.***

Tareas con distinto periodo de muestreo

¿Qué pasa si hay tareas con tiempos de ejecución mayor que otras? Lo único que se puede hacer es ejecutar una tarea cada n periodos de muestreo, con lo que su periodo será n × Ts. Si alguna tarea ha de ejecutarse con un periodo que no es múltiplo del periodo básico Ts,

entonces es necesario recurrir a un sistema Foreground/Background con varios temporizadores o a un sistema operativo de tiempo real.

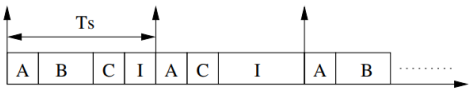
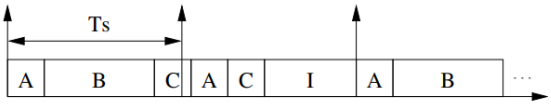


Figura 1.4: Temporización de tareas. La tarea B tiene un periodo = $2T_s$.

Tareas con tiempo de ejecución largo

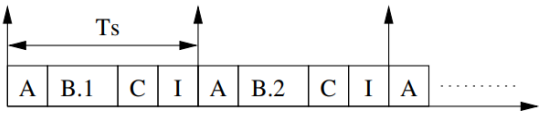
¿Y qué pasaría si el tiempo de ejecución de las tareas es mayor a Ts (time sample – periodo de muestreo)?



Como se puede observar, la tarea B tiene un periodo de 2Ts, pero tarda en ejecutarse más de la mitad del periodo de muestreo, dejando sin tiempo a la tarea C para terminar antes del final del periodo de muestreo.

Una primera solución puede consistir en cambiar el procesador por uno más rápido.

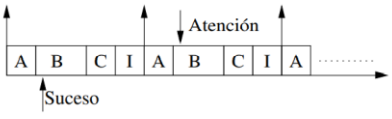
La otra solución es dividir la tarea B en dos mitades, el sistema será capaz de ejecutar las tareas A y C y la mitad de B en cada periodo de muestreo, tal como se ilustra en la figura:



Esta técnica no es tan fácil como parece, pues habrá que estimar el tiempo de ejecución de la tarea para averiguar por dónde se puede dividir. Esta estimación se complica cuando el tiempo de ejecución de la tarea es variable.

Latencia de las tareas en el bucle de scan

Se define como el tiempo máximo que transcurre entre un evento (interno o externo) y el comienzo de la ejecución de la tarea que lo procesa. En un bucle de scan, el caso más desfavorable se da cuando el suceso externo ocurre justo después de su comprobación por parte de la tarea. En este caso tiene que pasar todo un periodo de muestreo hasta que la tarea vuelve a ejecutarse y comprobar si ha ocurrido dicho suceso.



Por tanto, la latencia de un sistema basado en procesamiento secuencial es igual al tiempo máximo que tarda en ejecutarse el bucle de scan.

Ventajas e inconvenientes del bucle de scan:

Ventajas:

- Fácil implementación y depuración.

- Fácil compartir datos. No hay que sincronizar.
- Mayor eficiencia. No se pierde tiempo realizando cambio de contexto.

Desventajas:

- Latencia asociada al bucle scan.
- Dificultad implementar tareas que no sean múltiplo del periodo de muestro.
- Dificultad de escalar el programa o modificar, es decir agregar una nueva funcionalidad ya que si hago esto tengo que cambiar todo.
- Solo para sistemas sencillos. Si tengo que dividir muchas tareas para cumplir el tiempo de muestro resulta imposible.

Sistemas Foreground/Background

Hacen el uso de Interrupciones para esperar eventos externos asíncronos y disminuir Latencia. Se basa en tener dos tipos de tareas, unas de primer plano (foreground) y otras de segundo plano (Background).

- Un programa principal que se encarga de inicializar el sistema de interrupciones y luego entra en un bucle sin fin. Dentro de este bucle sin fin se ejecutarán las tareas de 1er plano: TareaA, TareaB . . . TareaI.
- Tareas de 2º plano, encargadas de gestionar algún suceso externo que provoca una interrupción. En la figura 1.9 estas tareas son las denominadas Tarea 1, Tarea 2 . . . Tarea n.

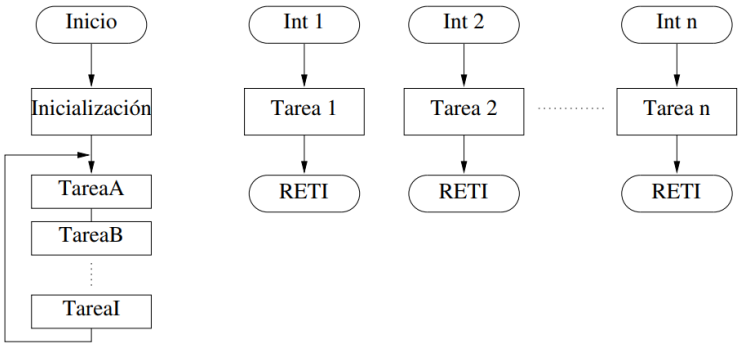


Figura 1.9: Ejemplo de sistema primer plano / segundo plano

Latencia en sistemas primer plano / segundo plano

En este tipo de sistema la latencia disminuye considerablemente debido a que ahora no tengo que esperar a mi bucle scan que termine cuando mi suceso ocurre después de la comprobación, si no que se atiende apenas llega una interrupción y se deja la tarea de primer plano que se estaba ejecutando, entonces ahora ***la latencia es el tiempo máximo que tarda una tarea de background*** ya que si llega otra interrupción tengo que esperar a que termine una tarea de background.

El problema que trae esto es el de los datos compartidos entre estos dos planos, ya que si la interrupción del temporizador se produce cuando se están copiando los argumentos de la tarea, se copian solamente los valores que estaban antes de la interrupción y a la hora de mostrar el resultado por pantalla, va a mostrar algo erróneo, hasta que se actualicen los datos antes de volver de llamar nuevamente a la tarea que se interrumpió.

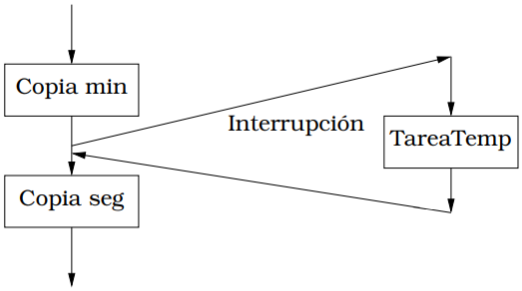


Figura 1.10: Incoherencia de datos.

La solución a esto es inhabilitar las interrupciones en lo que se llama a la ***zona critica*** que es donde se usa el recurso compartido, pero si el tiempo que tarda esta zona critica es grande aumenta la latencia de todo el sistema debido a que si llega una interrupción tengo que esperar a que termine de ejecutarse esta zona critica por lo tanto tiene que tardar el menor tiempo posible.