# Apuntes sistemas en tiempo real

Roberto Castellor Morant

24-02-17

# 1. Introducción a los Sistemas de Tiempo Real

## 1.1. Definición sistema en tiempo real

Cualquier sistema en el que el tiempo en el que se produce la salida es significativo. Esto generalmente es porque la entrada corresponde a algún movimiento en el mundo físico, y la salida esta relacionada con dicho movimiento. El intervalo entre el tiempo de entrada y el de salida debe ser lo suficientemente pequeño para una temporalidad aceptable.

Cualquier actividad o sistema de proceso de información que tiene que responder a un estimulo de entrada generado externamente con un retardo finito y especificado.

La correción de un sistema en tiempo real no depende sólo del resultado lógico de la computación, sino también del tiempo en el que se producen los resultados.

## 1.2. Tipos de sistemas en tiempo real

- Hard (estrictos) Son los sistemas en los que es absolutamente imperativo que las respuestas se produzcan dentro del tiempo límite especificado.
- Soft (no estrictos) Los tiempos de respuesta son importantes pero el sistema seguirá funcionando correctamente aunque los tiempos límite no se cumplan ocasionalmente.

## 1.3. Ejemplos de sistemas en tiempo real

## 1.4. Características de sistemas en tiempo real

Los sistemas en tiempo real tienen unas características especificas que los definen.

1. Funcionalidades en tiempo real
2. Control concurrente de sistemas separados
3. Programación de bajo nivel
4. Soporte a computación númerica
5. Grandes y complejos

6. Extremadamente fiables y seguros

7. Implementación eficiente y entorno de ejecución

# 2. Fiabilidad y tolerancia a fallos

Este tema, junto con el tema 3, trata de la producción de componentes de software fiables. Aunque se realizan consideraciones sobre la prevención de fallos, la atención se dedica principalmente a la tolerancia a fallos. Se condirean las técnicas de recuperación de errores hacia adelante y hacia atrás. El manejo de excepciones se estudia en el siguiente tema.

Causas que pueden propiciar el fallo de un sistema de tiempo real:

1. Especificación inadecuada.

2. Defectos provocados por errores de diseño.

3. Defectos provocados por fallos en componentes del procesador.

4. Defectos provocados por interferencias transitorias o permanentes en el subsistema de comunicaciones.

Los errores relacionados con el diseño o la especificación son dificiles de preveer mientras que los errores provocados por fallos son en cierto modo predecibles. Los lenguajes de programación de tiempo real tienen que ser altamente fiables.

## 2.1. Fiabilidad, fallos y defectos

Fiabilidad: una medida del éxito con el que el sistema se ajusta a alguna especificación definitiva de su comportamiento.

Fallo del sistema: Cuando el comportamiento de un sistema se desvía del especificado para él, se dice que es un fallo.

Se pueden distinguir tres tipos de fallos:

1. Fallos transitorios, Un fallo transitorio comienza en un instante de tiempo concreto, se mantiene en el sistema durante algún periodo, y luego desaparece. Ejemplos de este tipo de fallos se dan en componentes hardware en los que se produce una reacción adversa a una interferencia externa, como la producida por un campo eléctrico o por radioactividad. Después de que la perturbación desaparece, lo hace también el fallo (aunque no necesariamente el error inducido). Muchos de los fallos en los sistemas de comunicación son transitorios.

2. Fallos permanentes Los fallos permanentes comienzan en un instante determinado y permanecen en el sistema hasta que son reparados; es el caso, por ejemplo, de un cable roto o de un error de diseño de software.

3. Fallos intermitentes Son fallos transitorios que ocurren de vez en cuando. Un ejemplo es un componente hardware sensible al calor, que funciona durante un rato, deja de funcionar, se enfría, y entonces comienza a funcionar de nuevo

## 2.2. Modos de fallo

Se pueden identificar dos dominios generales de modos de fallo:

1. Fallos de valor, el valor asociado con el servicio es erróneo.
2. Fallos de tiempo, el servicio se completa a destiempo.

Las combinaciones de fallos de valor y de tiempo se denominan fallos arbitrarios.

Un fallo de valor fuera del rango esperado para el servicio se denomina error de límites, son fallos fácilmente reconocibles.

Los fallos en el dominio del tiempo se pueden englobar en:

1. Demasiado pronto, el servicio es entregado antes de lo requerido.
2. Demasiado tarde, el servicio se entrega después de lo requerido, se puede hablar de error de prestaciones.
3. Infinitamente tarde, el servicio nunca es entregado, fallo de omisión.

Esta clasificación se puede ampliar con fallos de encargo o improvisación cuando el servicio es entregado sin ser esperado.

Dada la clasificación de fallos se puede definir algunas suposiciones respecto al modo en que los sistemas pueden fallar:

1. Fallo descontrolado, un sistema que produce fallos arbitrarios tanto en el dominio del valor como del tiempo
2. Fallo de retraso, un sistema produce servicios correctos en el dominio del valor pero no en el del tiempo.
3. Fallo de silencio, cuando el sistema falla bruscamente sin haber tenido fallos de valor o de tiempo, a partir del fallo todos los servicios subsiguientes también sufren fallos de omisión.
4. Fallo de parada, un sistema que cumple los requisitos de un fallo de silencio pero permite que otros sistemas detectan que ha entrado en el estado de fallo
5. Fallo controlado, un sistema falla de una forma especificada y controlada.
6. Sin fallos, un sistema produce los servicios correctos, tanto en el dominio del valor como del tiempo.

# 3. Excepciones y manejo de excepciones

Se continúa con el estudio realizado en el tema anterior sobre la producción de componentes de software fiables, centrándose este tema en el manejo de las excepciones, estudiándose tanto los modelos de terminación como los de reanudación.

Los mecanismos de manejo de excepciones tienen que cumplir estos requisitos:

1. El mecanismo debe ser facil de comprender y utilizar.

2. El código de manejo de excepciones no debe oscurecer el código normal del programa.

3. El mecanismo debe diseñarse de forma que no suponga una sobrecarga en la ejecución. Solamente sera aceptable cierta pequeña sobrecarga cuando sea primordial la rapidez de recuperación al ocurrir una excepción.

4. El mecanismo debe permitir un tratamiento uniforme de las excepciones tanto por el entorno como por el programa.

5. El mecanismo debe permitir la programación de acciones de recuperación.

Existen métodos de control de excepciones en lenguajes tipo C implementados con valores de retorno inusuales, normalmente distintos de cero. Otro método de control de excepciones en este tipo de lenguajes es mediante la bifurcación forzada, por ejemplo el uso de goto en C aunque este tipo de control de excepciones generan un código más oscuro.

Actualmente en los lenguajes de programación modernos se tiende a incluir funcionalidades de manejo de exepciones directamente en el lenguaje, lo que provee un mecanismo más estructurado.

Según el retraso en la detección del error las excepciones se provocan de forma síncrona o asíncrona, la excepción síncrona se genera como respuesta a un bloque de código, la excepción asíncrona se genera un tiempo despues de que ocurra la operación que da lugar a la aparición del error.

Existen cuatro tipos de excepciones:

1. Detectada por el entorno y generada síncronamente, por ejemplo una división por cero.

2. Detectada por la aplicación y generada síncronamente, por ejemplo un fallo en un aserto.

3. Detectada por el entorno y generada asíncronamente, por ejemplo un fallo de alimentación o un mecanismo de monitorización vital.

4. Detectada por la aplicación y generada asíncronamente, por ejemplo cuando un proceso verifica una condición que ocasionará que un proceso no cumpla los plazos o no termine correctamente.

Existen diferentes formas de declarar las excepciones síncronas:

1. Mediante un nombre constante que necesita ser declarado explícitamente.

2. Mediante un objeto de cierto tipo que podrá o no ser declarado explícitamente.

Ada precisa que las excepciones se declaren como constantes, por ejemplo las excepciones que puede generar el entorno de ejecución se declaran en el paquete Standard.

```
package Standard is
....
Constraint_Error : exception;
Program_Error : exception;
Storage_Error : exception;
Tasking_Error : exception;
...
end Standard
```

Java y C++ dan una visión de las excepciones más orientada al objeto. En java las excepciones son instancias de una subclase de la clase Throwable y podran ser lanzadas tanto por el sistema de ejecución como por la aplicación. En C++ se pueden lanzar excepciones de cualquier tipo de objeto sin declararlas previamente.

Se considera que un bloque esta vigilado (guarded) cuando se ha definido un manejador de excepciones para el, por ejemplo en Java se considerará vigilado el bloque en una estructura try ... catch. Al definir los bloques podemos incurrir en un problema de granularidad inadecuada cuando el bloque es demasiado grande y no podemos definir en que instrucciones se produce la excepción. Una solución a este problema es disminuir el tamaño del bloque y/o anidarlo. Otra alternativa es crear procedimientos con manejadores para cada uno de los bloques anidados. La mejor aproximación es permitir el paso de parametros junto a las excepciones.

### 3.1. Propagación de excepciones

Cuando no hay un manejador de excepciones en un bloque de código y se produce una excepción hay dos formas de proceder:

- Entender la ausencia de manejador como un error de programación y notificarlo en tiempo de compilación.
- La segunda vía es buscar manejadores en el pasado de la cadena de invocación en tiempo de ejecución, esto se conoce como propagación de excepciones.

## 4. Programación concurrente

En este tema se introuce la noción de proceso, tarea y hebra o hilo y revisa los modelos que utilizan los diseñadores de lenguajes y de sistemas operativos. El término tarea se utiliza genéricamente para representar una actividad concurrente. Se aprovecha también este tema pra estudiar la distribución de tareas cuando se dispone de multiprocesadores o de sistemas distribuidos. Dejándose para los siguientes dos temas la comunicación entre tareas.

## 5. Sincronización y comunicación basada en variables compartidas

Este tema, junto con el siguiente, se centra en el estudio de la comunicación entre tareas. En concreto en este primer tema se escriben los métodos de varia-

bles compartidas, incluyendo la utilización de semáforos, monitories, variables compartidas y objetos protegidos.

# 6. Sincronización y comunicación basada en mensajes

Este tema es la continuación del anterior y resalta la importancia que tienen en los lenguajes modernos los métodos basados en mensajes para la comunicación y sincronización.

# 7. Acciones atómicas, Tareas Concurrentes y Fiabilidad

En este tema se amplian las discusiones iniciales sobre tolerancia a fallos describiendo con cuánta fiabilidad puede ser programada la cooperación entre procesos. Para esta discusión es fundamental la noción de acción atómica y las técnicas de manejo de eventos asíncronos.

# 8. Control de recursos

Como continuación del tema anterior, en este tema se tratan los procesos competitivos. Un tema importante en este caso es la distinción entre sincronización condicional y sincronización evitable en el modelo de concurrencia.

## 8.1. Interbloqueo o bloqueo mutuo

Tambien conocido como deadlock se trata del bloqueo permanente de un conjunto de procesos o hilos de ejecucion que compiten o se comunican entre ellos en un sistema concurrente. No existe una solución general para los interbloqueos.

Es posible representar bloqueos mutuos mediante grafos dirigidos, el proceso es representado por un cuadrado y el recurso por un circulo. Cuando un proceso solicita un recurso se dibuja una flecha dirigida desde el el proceso al recurso, cuando el recurso esta asignado a un proceso la flecha esta dirigida desde el recurso al proceso.

Existen cuatro condiciones necesarias para que se de un interbloqueo, estas son:

- Condición de exclusión mutua, existe un recurso compartido por los procesos al que solo puede acceder un proceso simultaneamente.
- Condición de retención y espera, al menos un proceso P ha adquirido un recurso R y lo retiene mientras espera un recurso R2 que ha sido asignado a otro proceso.
- Condición de no expropiación, los recursos no pueden ser expropiados por otros procesos, tienen que ser liberados por sus propietarios.

- Condición de espera circular, dado un conjunto de proceso P0..PM, P0 esta esperando un recurso adquirido por P1, que esta esperando un recurso adquirido por P2, ..., que esta esperando un recurso adquirido por PM, que esta esperando un recurso adquirido por P0. Esta condición implica la condición de retención y espera.

# 9. Capacidades de tiempo real

Los requisitos temporales constituyen la característica diferenciadora de los sistemas de tiempo real. En este tema se presenta una visión de estos requisitos y de las funcionalidades del lenguaje y estrategias de implemntación que se utilizan para satisfacerlos. Los sistemas de tiempo real estrictos tienen restricciones de tiempo que deben ser satisfechas, los sistemas no estrictos fallan a veces a la hora de cumplir dichas restricciones adecuadamente. Los dos casos se consideran en el contexto de la planificación con tiempos límite.

# 10. Planificación

Tras el tema anterior de capacidades de tiempo real, hay que incluir estas capacidades en la planificación de tareas, introduciéndose también la noción de prioridad junto con el análisis de la planificabilidad para sistemas con desalojo basado en prioridad.

Vamos a barajar tres modelos de planificación:

- Modelo simple
- Ejecutivo cíclico
- Planificación basada en procesos

## 10.1. Modelo simple

Tenemos un modelo que permite describir algunos esquemas de planificación estándar:

- La aplicación está compuesta por un conjunto fijo de procesos
- Los procesos son periódicos, con periodos conocidos
- Los procesos son independientes entre sí
- El instante crítico es cuando todos los procesos son ejecutados a la vez
- Las sobrecargas del sistema, tiempos de cambio de contexto y demás se suponen con coste cero
- Los tiempos límite de los procesos son iguales a sus periodos
- Los procesos tienen tiempo de ejecución constante en el peor caso

Para definir el modelo de proceso simple definimos las siguientes variables:

| Notación | Descripción |
|---|---|
| B | Tiempo de bloqueo del proceso en el peor caso |
| C | Tiempo de ejecución del proceso en el peor caso (WCET) |
| D | Tiempo límite del proceso |
| I | Tiempo de interferencia del proceso |
| J | Fluctuación en la ejecución del proceso |
| N | Número de procesos en el sistema |
| P | Prioridad asignada al proceso |
| R | Tiempo de respuesta del proceso en el peor caso |
| T | Tiempo mínimo entre ejecuciones del proceso (periodo) |
| Y | Utilización de cada proceso (C/T) |
| a-z | Nombre del proceso |

### 10.2. Ejecutivo cíclico

Una forma común de la implementación de sistemas de tiempo real es el uso de un ejecutivo cíclico.

Existe una tabla de llamadas a procedimiento con un ciclo principal y ciclos secundarios de duración fija.

Las propiedades del ejecutivo cíclico son:

- Los procesos no existen en tiempo de ejecución, solamente existe el proceso principal
- Los procedimientos comparten un espacio de direcciones, pueden compartir datos sin necesidad de protección
- Los periodos deben ser múltiplos del tiempo de ciclo secundario

El ejecutivo cíclico presenta los siguientes inconvenientes:

- Dificultad para incorporar procesos esporádicos
- Dificultad para incorporar procesos con periodos grandes, el tiempo del ciclo mayor es el único que se puede usar sin replanificación
- Dificultad para construir el ejecutivo cíclico
- Procesos con tiempo notable tendrán que ser divididos en procedimientos de tamaño fijo, es propenso a errores

## 11. Programación de bajo nivel

Un requisito importante de muchos sistemas de tiempo real es que incorporan dispositivos externos que deben ser programados (controlados) como una parte del software de la aplicación. Esta programación a bajo nivel no concuerda con la aproximación abstracta para la producción de software que caracteriza a la ingeniería del software. En este tema se considera las formas en que las funcionalidades de bajo nivel pueden ser incorporadas con exito en los lenguajes de alto nivel.