Sistemas Operativos de Tiempo Real

Ing. José Roberto Vignoni

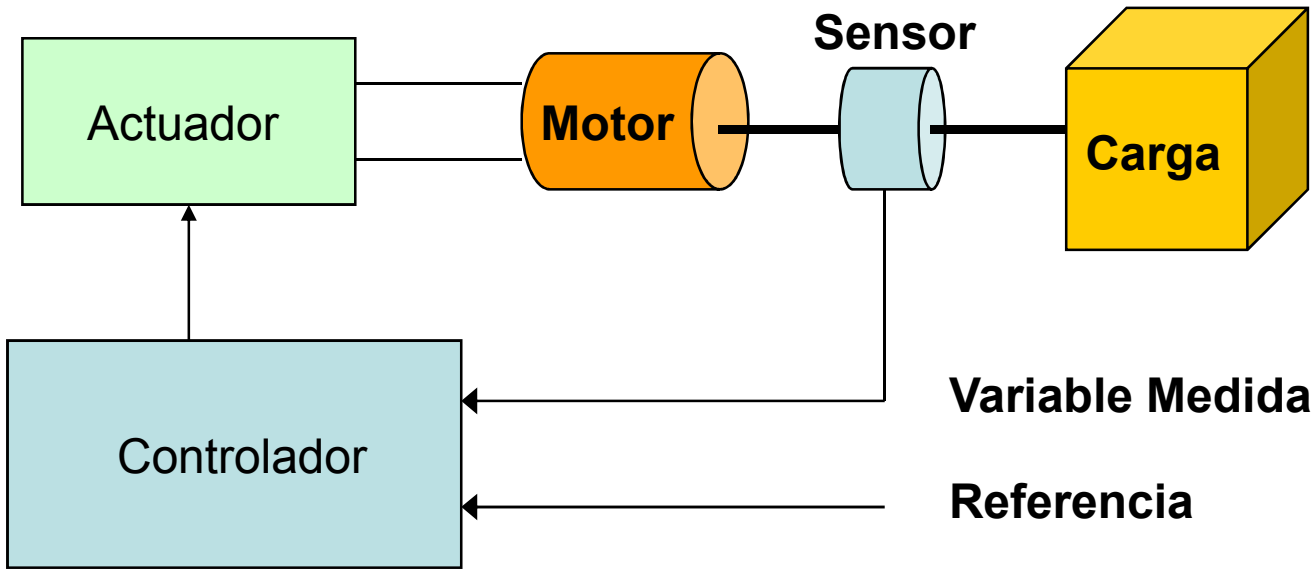
Año 2004

- Un **sistema de tiempo real** es un sistema informático que:
- Interacciona repetidamente con su entorno físico
- Responde a los estímulos que recibe del mismo dentro de un plazo de tiempo determinado
- Podemos decir que el procesamiento en tiempo real, es un tipo de procesamiento en el que la exactitud del sistema no depende solo del resultado lógico de un cálculo, sino también del instante en que se produzca este resultado.

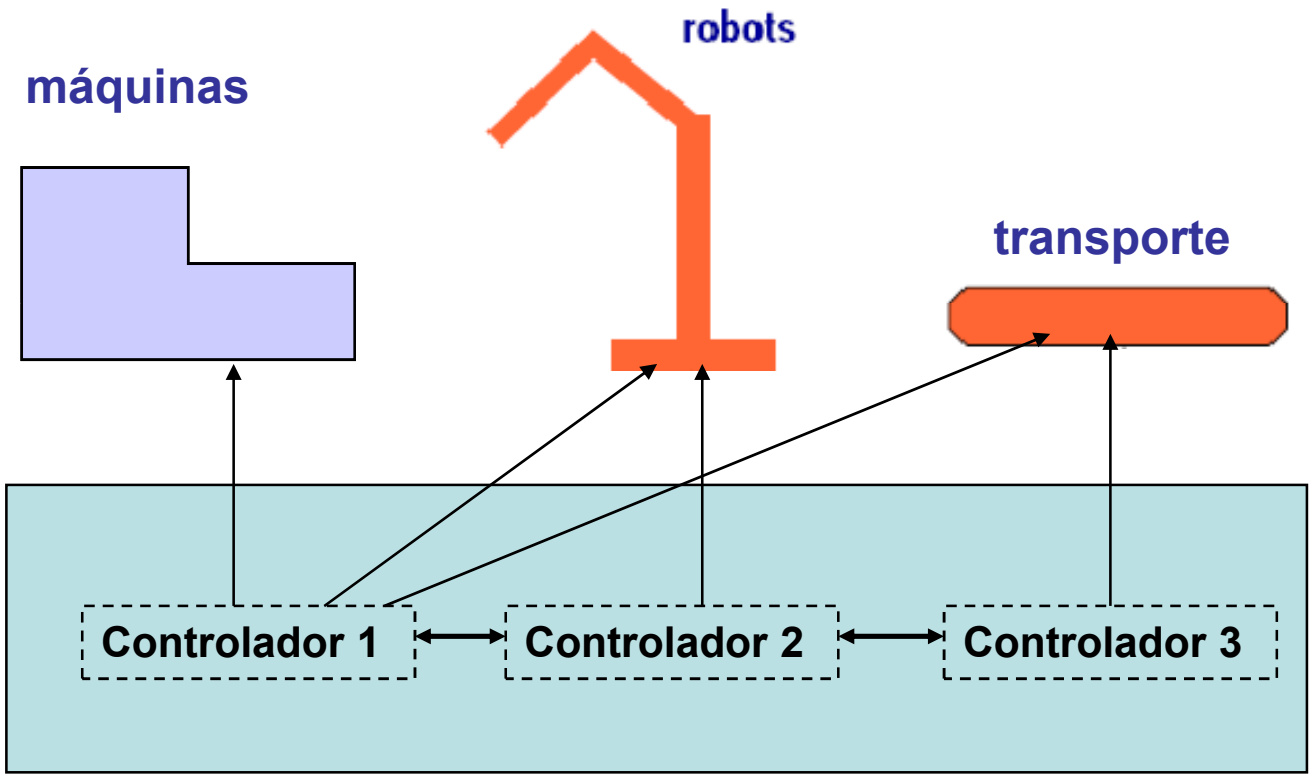
Ejemplos de aplicación

- Aviónica
- Control de tráfico aéreo
- Control de trenes
- Control de automóviles
- Telecomunicaciones
- Producción y distribución de energía eléctrica
- Electrónica de consumo
- Sistemas multimedia

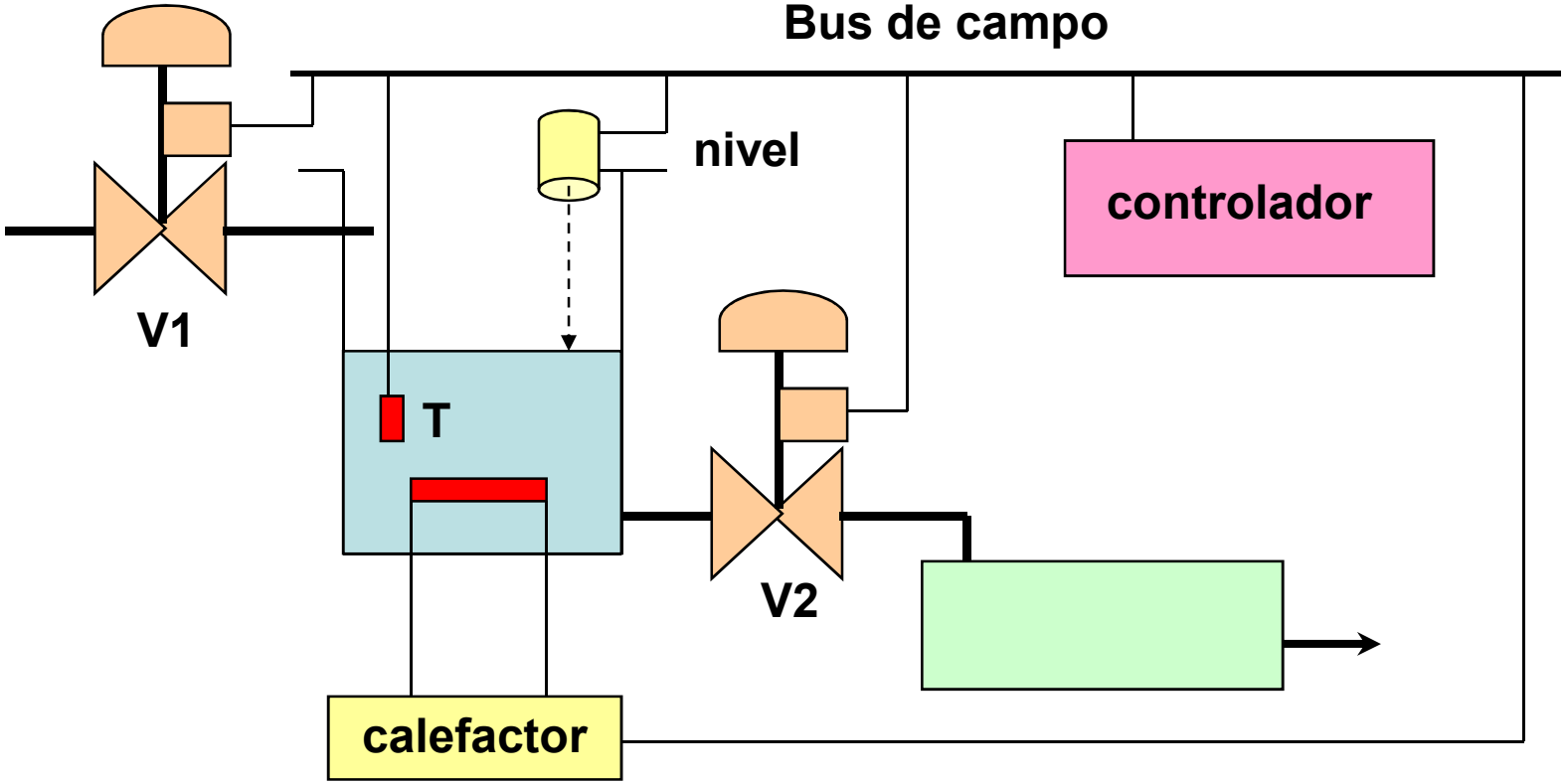
Control de Posición



Control de Fabricación



Control de Proceso



Sistemas embebidos o empotrados (embedded systems)

- Son las aplicaciones en que los sistemas de tiempo real realizan funciones de control como componentes de otros sistemas.

Ejemplos	Características
-Automóviles -Electrónica de consumo: TV, celular, etc. -Electrodomésticos	-El controlador y/o el SO no es accesible -Los recursos son limitados

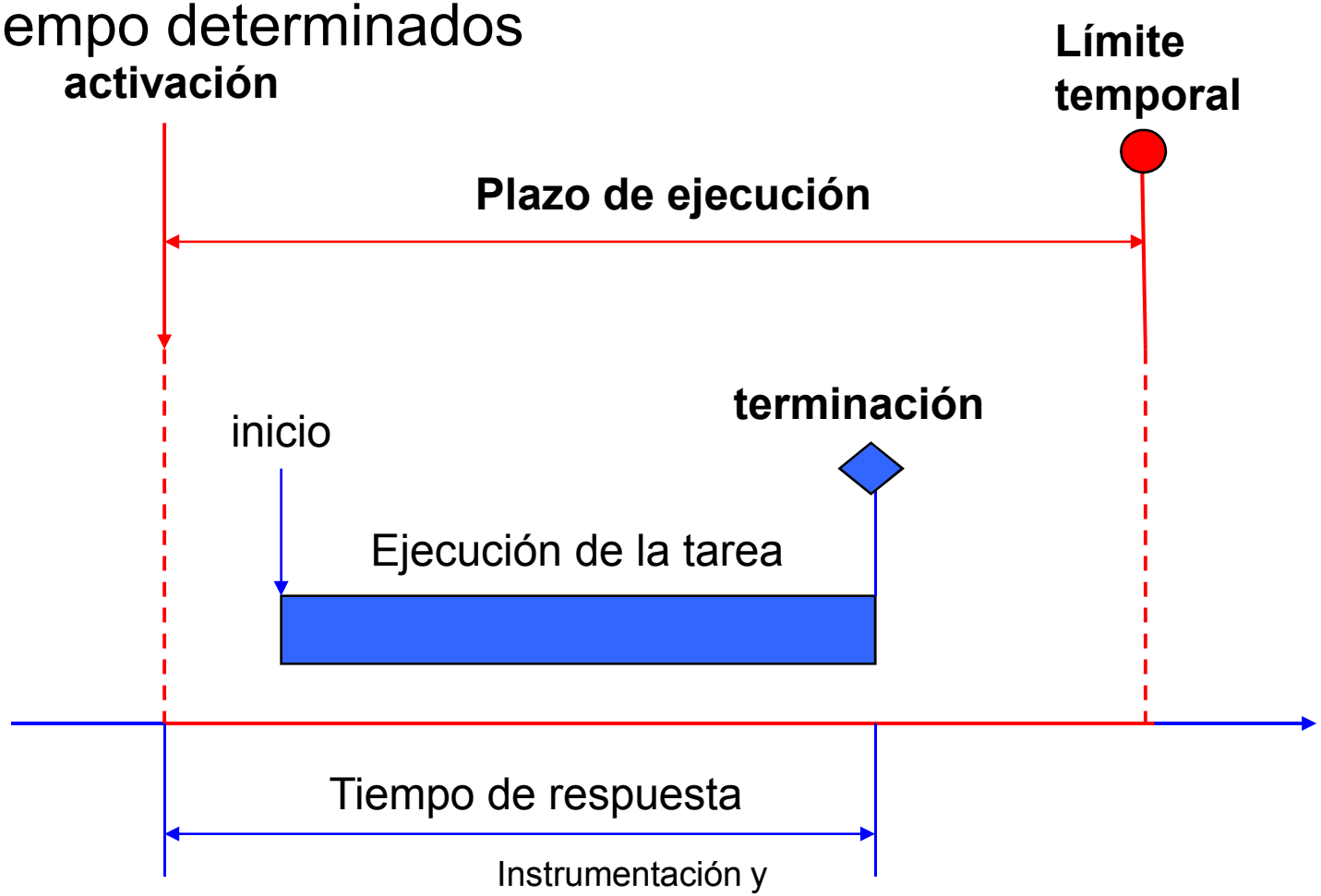
- Adquisición de datos
  - Variables
    - Cada variable pertenece al entorno de control de un sistema o subsistema.
    - Fuera de su entorno se puede observar pero no cambiar.
    - La imagen (valor) de una variable en un STR tiene un intervalo de validez.
    - Las secuencias de tomas de datos pueden estar disparadas por tiempo o por sucesos.

Interacción con las personas

- Interfaz con el usuario (HMI)
  - presentación de datos
  - presentación de alarmas
  - presentación de tendencias
  - registro de datos
  - generación de informes

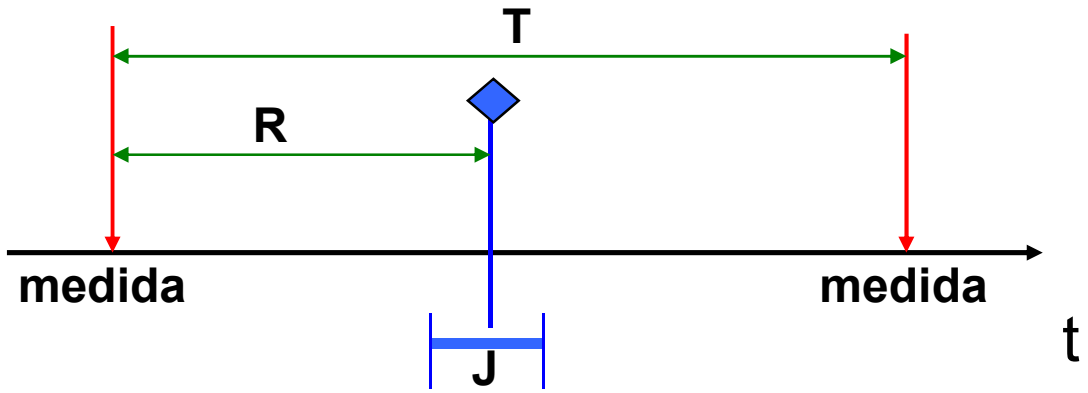
Restricciones temporales

Las tareas en un SOTR se ejecutan, generalmente en forma repetitiva siempre dentro de intervalos de tiempo determinados



El controlador muestrea periódicamente la variable controlada, la compara con la referencia (set point), calcula la variable de control, utilizando el algoritmo de control.

El controlador tarda un cierto tiempo de respuesta en calcular la acción de control.



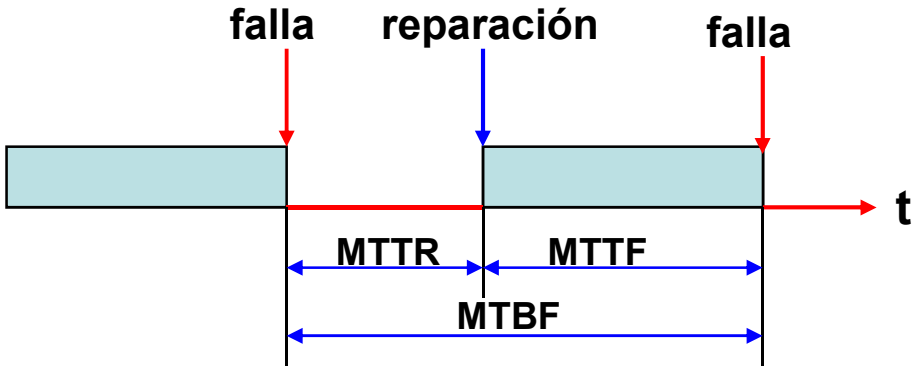
J es la variación (Jitter) en el tiempo de respuesta:

$$J = R_{max} - R_{min}$$

Fiabilidad y seguridad

- Fiabilidad
  - Es la posibilidad de que el SO proporcione el servicio especificado.
  - Tiempo medio entre fallas o averías MTTF = $1/\lambda$
    - Donde λ es una tasa de fallas constante en fallas por hora
    - Un sistema con MTTF> 10^9 se denomina ultrafiable
- Seguridad
  - Tipos de averías: malignas y benignas
    - Una avería maligna tiene un coste muy superior a la utilidad del sistema
  - Sistemas críticos
    - Deben ser ultrafiables respecto a las averías malignas
    - En muchos casos se exige una certificación de seguridad efectuada por un organismo independiente

- Mantenibilidad
  - Medida del tiempo necesario para reparar una avería benigna.
  - Tiempo medio de reparación: MTTR = 1/μ
    - Para una tasa de reparación de μ reparaciones / hora
- Disponibilidad
  - Fracción de tiempo en que el sistema está disponible



Clases de Sistemas de Tiempo Real

Según las propiedades del sistema controlado

- sistemas críticos y sistemas no críticos
- sistemas con parada segura y sistemas con degradación aceptable

Según las propiedades del sistema de tiempo real

- sistemas con tiempo de respuesta garantizado y sistemas que hacen lo que pueden
- sistemas con recursos adecuados y sistemas con recursos inadecuados
- sistemas disparados por tiempo y sistemas disparados por eventos

SOTR Críticos y no críticos

Se distinguen por sus requisitos temporales y de fiabilidad

SOTR Críticos
(hard real time systems)

- Plazo de respuesta estricto
- Comportamiento temporal determinado por el entorno
- Comportamiento en sobrecargas predecible
- Requisitos de seguridad críticos
- Redundancia activa
- Volumen de datos reducido

SOTR No Críticos
(soft real time systems

- Plazo de respuesta flexible
- Comportamiento temporal determinado por el computador
- Comportamiento en sobrecargas degradado
- Requisitos de seguridad no críticos
- Recuperación de fallos
- Gran volumen de datos

Sistemas con parada segura y sistemas con degradación aceptable

Se distinguen por su comportamiento en caso de falla

Sistemas con parada segura (fail-safe)

- Detención en estado seguro
- Probabilidad de detección de fallos elevada

Sistemas con degradación aceptable (fail-soft)

- Funcionamiento con pérdida parcial de funcionalidad o prestaciones
- También hay sistemas con tolerancia de fallos completa (fail operational)

Sistemas con respuesta garantizada y sistemas que hacen lo que pueden

Se distinguen por su grado de determinismo temporal

Sistemas con respuesta garantizada (guaranteed response systems)

- Comportamiento temporal garantizado analíticamente
- Hace falta caracterizar con precisión la carga máxima y los posibles fallos

Sistemas que hacen lo que pueden (best-effort systems)

- Comportamiento temporal de tipo “lo mejor que se pueda”
- No se hace una caracterización precisa de carga y fallos
- Sólo sirve para sistemas no críticos

Sistemas con recursos adecuados e inadecuados

Se distinguen por la cantidad de recursos disponibles

Sistemas con recursos adecuados (resource adequate systems)

• Diseño con suficientes recursos para garantizar el comportamiento temporal con máxima carga y en caso de fallos

Sistemas con recursos inadecuados (resource inadequate systems)

• Diseño con recursos "razonables" desde un punto de vista económico
• Sólo sirve para sistemas no críticos

Sistemas disparados por tiempo y por sucesos

Se distinguen por la forma de arrancar la ejecución de sus actividades

Sistemas disparados por Sucesos (event-triggered systems)

- Arranque cuando se produce un suceso de cambio de estado
- Mecanismo básico: interrupciones

Sistemas disparados por tiempo (time - triggered systems)

- Arranque en instantes de tiempo predeterminados
- Mecanismo básico: reloj

Resumen

- Los sistemas operativos convencionales no son adecuados para realizar tareas de tiempo real
  - no tienen un comportamiento determinista
  - no permiten garantizar los tiempos de respuesta
- Un sistema operativo de tiempo real debe soportar

– **concurrencia**: procesos ligeros (threads) con memoria común

– **temporización**: medida de tiempos y ejecución periódica

– **planificación**: prioridades fijas con expropiación, acceso a recursos con protocolos de herencia de prioridad

– **dispositivos de E/S**: acceso a recursos de hardware e interrupciones

Lenguajes de Programación para SOTR

Un lenguaje de programación de sistemas de tiempo real debe facilitar la realización de sistemas

– concurrentes,

– fiables,

– con un comportamiento temporal analizable.

Hay tres clases de lenguajes de interés para SOTR:

– Lenguajes ensambladores

» Flexibles y eficientes, pero costosos y poco fiables

– Lenguajes secuenciales (Fortran, Pascal, C, ...)

» Necesitan servicios especiales para concurrencia y tiempo real

– Lenguajes concurrentes (Modula, Ada, ...)

» Concurrencia y tiempo real incluidos en el lenguaje

C

- Es un lenguaje muy utilizado para programación de sistemas
  – estructurado, con bloques
  – sin tipado fuerte
  – muy flexible (pero a veces poco seguro)

- No tiene integrada la concurrencia ni el tiempo real
  – se consigue invocando servicios del sistema operativo de forma explícita
  - No facilita la descomposición en módulos ni la programación con objetos
  – se puede hacer con **C++**
  (una extensión de C para programar con objetos)

ADA

- Es un lenguaje diseñado específicamente para sistemas de tiempo real
  - concurrencia
  - tiempo real
  - acceso al hardware e interrupciones
- Es un lenguaje imperativo, descendiente de Pascal
  - estructura en bloques
  - fuertemente tipado
  - paquetes (módulos) y esquemas genéricos
  - extensión de tipos con herencia
  - biblioteca jerárquica
  - interfaces normalizadas con otros lenguajes (C, Fortran)

POSIX

- Es un conjunto de normas IEEE/ISO que definen **interfaces de sistemas operativos**
- Permiten desarrollar software portátil y reutilizable
- (**P**ortable **O**perating **S**ystem **I**nterface) + **X**
- Las normas definen **servicios** que se pueden incluir o no en un sistema operativo particular
- Además se definen **perfiles de aplicación** con conjuntos de servicios estándar
- Hay interfaces para C, Ada, y otros lenguajes

El tiempo en SOTR

- Acceso al **tiempo real**
  – leer el paso del tiempo en relojes
  – retrasar la ejecución de los procesos durante un tiempo
  – definir límites temporales para la ocurrencia de un suceso (time-outs)
  – ejecutar acciones en determinados instantes
- Representación de los **requisitos temporales**
  – períodos de activación
  – plazos de ejecución

Relojes

- Los relojes son módulos de hardware y software que permiten medir el tiempo real.
- Pueden ser internos o externos
- **Características importantes**:
  - Características estáticas (representación del tiempo)
    - » Resolución
    - » Intervalo de valores
  - Características dinámicas
    - » Granularidad
    - » Exactitud
    - » Estabilidad

Relojes de tiempo de ejecución

- Permiten medir el tiempo de ejecución que ha consumido

  una actividad (hebra de sistema operativo o tarea de Ada) desde su inicio
- Se pueden usar para medir el tiempo de ejecución de un

  segmento de código

  – problema: ¿cómo medir el tiempo de ejecución en el peor caso? (WCET)

  » los programas no se ejecutan siempre igual

  » algunos componentes de la arquitectura introducen variaciones importantes (caches, segmentación, ejecución expropiativa)

  » soluciones: modelos, procesadores más sencillos

Retardos

- Un **retardo** suspende la ejecución de una tarea durante un cierto tiempo
- Hay dos tipos
  - **Retardo relativo**: la ejecución se suspende durante un intervalo de tiempo relativo al instante actual
  - **Retardo absoluto**: la ejecución se suspende hasta que se llegue a un instante determinado de tiempo absoluto

Temporizadores

- Un temporizador es un mecanismo (de hardware y software) que permite avisar que ha transcurrido un cierto tiempo
  – una sola vez desde que se *arma*
  – periódicamente

- El mecanismo por el cual se avisa depende del sistema operativo y del lenguaje de programación
  – POSIX: señales
  – Ada: mecanismos del lenguaje (selección temporizada)

Requisitos Temporales

- Los marcos temporales suelen ir asociados a tareas o procesos
- Generalmente se trata de
  – Ejecutar **tareas periódicas**
  – Ejecutar **tareas esporádicas** cuando ocurren los sucesos correspondientes
  – Completar la ejecución de todas las tareas dentro de su **plazo de respuesta**
- A veces se exige que la entrada o salida de una tarea se efectúe a intervalos regulares
- La desviación se llama fluctuación o ***jitter***

Análisis Temporal

- Si el sistema cumple determinadas propiedades estructurales (modelo de tareas) se puede analizar su comportamiento temporal
  – condiciones para cumplir los plazos
  – cálculo del tiempo de respuesta

- Los métodos de análisis temporal están estrechamente relacionados con la planificación de las tareas
  – el método de planificación debe asegurar un comportamiento temporal *previsible* y *analizable*

Fallos Temporales

- Una tarea puede incumplir su plazo por varias razones, por ejemplo:
- El **tiempo de cómputo** no está bien calculado
- El **análisis** de tiempos de respuesta no es realista
- Las **herramientas** de análisis contienen errores
- No se cumplen las **hipótesis** de diseño
  (por ejemplo, separación mínima entre eventos)
- En estos casos hay que **detectar** los fallos
- Si el sistema es crítico, debe **recuperarse**

Resumen

- Hay cuatro aspectos importantes relacionados con el tratamiento del tiempo
  – medida del tiempo mediante relojes
  – retardos y temporizadores
  – limitación del tiempo de espera
  – especificación de requisitos temporales
- Los atributos más importantes de un marco temporal (generalmente asociado a una tarea) son:
  – esquema de activación (periódico, esporádico)
  – plazo de terminación
  – latencia de activación
  – tiempo de cómputo máximo

Criticidad

Una tarea de tiempo real puede ser

- **Crítica** (hard) : No se puede admitir que se sobrepase el

  plazo de respuesta especificado ni una sola vez
- **Acrítica** (soft) : Es admisible que se sobrepase el plazo

  ocasionalmente
- **Firme** (firm) : El plazo no es crítico, pero una respuesta

  tardía no sirve para nada
- **Interactiva** : No se especifican plazos de respuesta, sino tiempos de respuesta medios

Fallos temporales

Una tarea puede incumplir su plazo por varias razones, por ejemplo:

- El **tiempo de cómputo** no está bien calculado
- El **análisis** de tiempos de respuesta no es realista
- Las **herramientas** de análisis contienen errores
- No se cumplen las **hipótesis** de diseño
  (por ejemplo, separación mínima entre eventos)
- En estos casos hay que **detectar** los fallos
- Si el sistema es crítico, debe **recuperarse**

Planificación en tiempo real

En el estudio de los algoritmos de planificación de tiempo real, se observa que los métodos de planificación dependen de:

– Si el sistema lleva a cabo un análisis de planificación

– En caso afirmativo si se realiza en forma estática o dinámica

– Si el resultado del análisis genera un plan con respecto al cual se expiden las tareas durante la ejecución.

En base a las consideraciones anteriores se pueden identificar las siguientes clases de algoritmos:

- Métodos con tablas estáticas:

  Realizan un análisis estático de las planificaciones posibles. El resultado del análisis es un plan que determina, cuando debe comenzar o terminar la ejecución de una tarea. Es aplicable a tareas periódicas. Los datos iniciales son:

  – Tiempo periódico de llegada
  – Tiempo de ejecución
  – Plazo periódico de finalización
  – Prioridad relativa de cada tarea

– El planificador intenta trazar un plan que le permita cumplir las exigencias de todas las tareas periódicas. Es un método predecible e inflexible, ya que cualquier cambio de exigencia en una tarea, requiere un nuevo plan.

- Métodos apropiativos con prioridades estáticas:

  También se realiza un análisis estático, pero no se traza ningún plan. En cambio, se usa dicho análisis para asignar prioridades a tareas, con lo que se puede usar un planificador apropiativo con prioridades convencional. En este caso la asignación de prioridades se encuentra relacionada con las restricciones de tiempo asociadas a cada tarea.

Modelo de Tareas

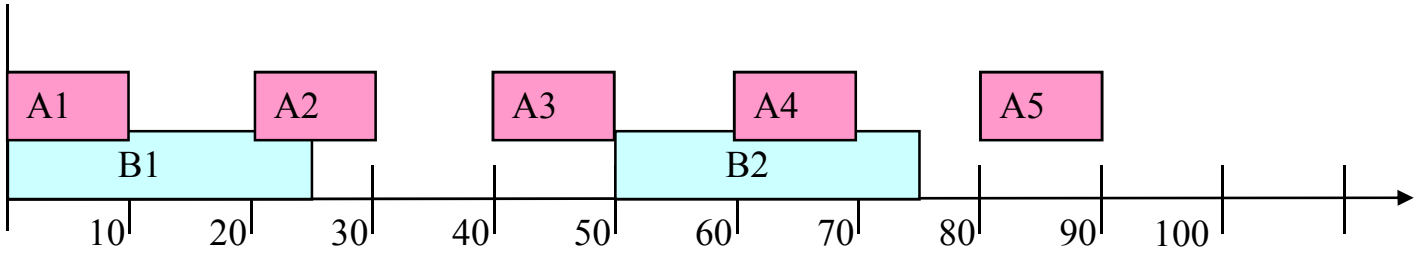
Consideraremos un **modelo simple**:

- El conjunto de tareas es **estático**
- Todas las tareas son **periódicas**
- Las tareas son **independientes** unas de otras
- Los **plazos** de respuesta de todas las tareas son iguales a los **períodos** respectivos
- El **tiempo de ejecución máximo** de cada tarea es conocido
- Las operaciones del núcleo de multiprogramación son **instantáneas**

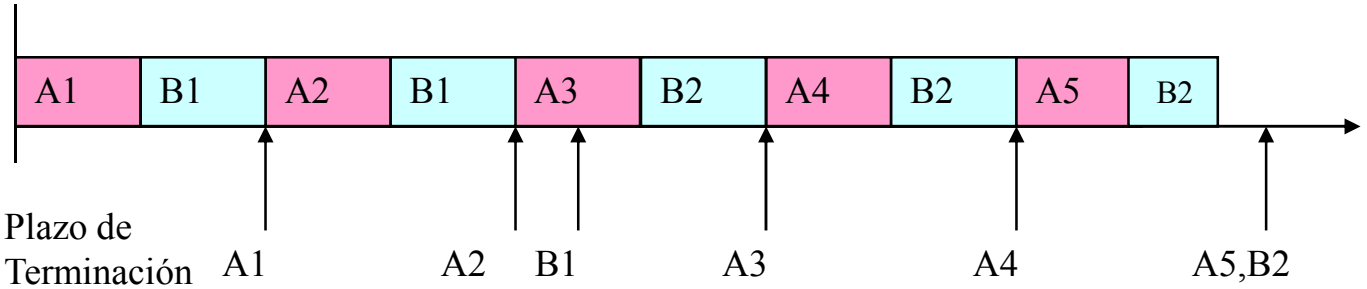
Ejemplo

Consideremos un ejemplo de planificación periódica de tareas con plazo de terminación. Dados dos sensores A y B, el plazo para tomar datos del sensor A es cada 20 ms y del sensor B cada 50 ms. Se tarda 10 ms, incluida la sobrecarga del SO para procesar los datos de A y 25 ms los datos de B.

	Lleg.	Ejec.	Plazo
A1	0	10	20
A2	20	10	40
A3	40	10	60
A4	60	10	80
A5	80	10	100
B1	0	25	50
B2	50	25	100

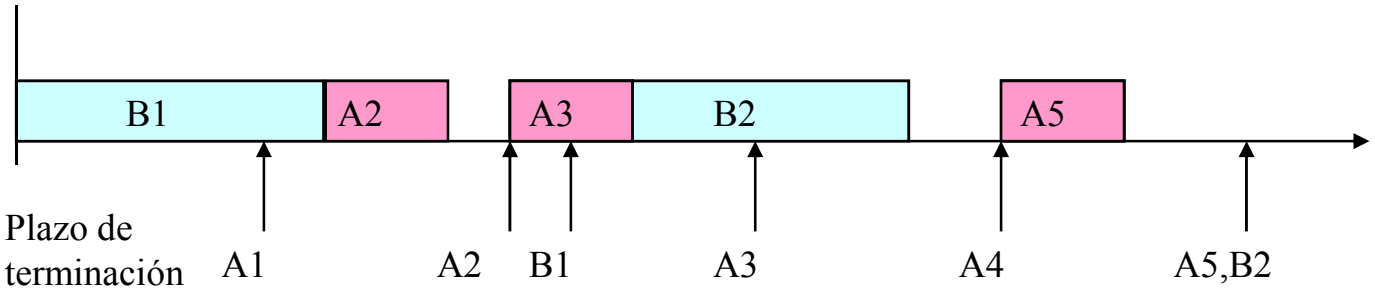


Consideremos el caso de planificación por prioridades con A de mayor prioridad que B



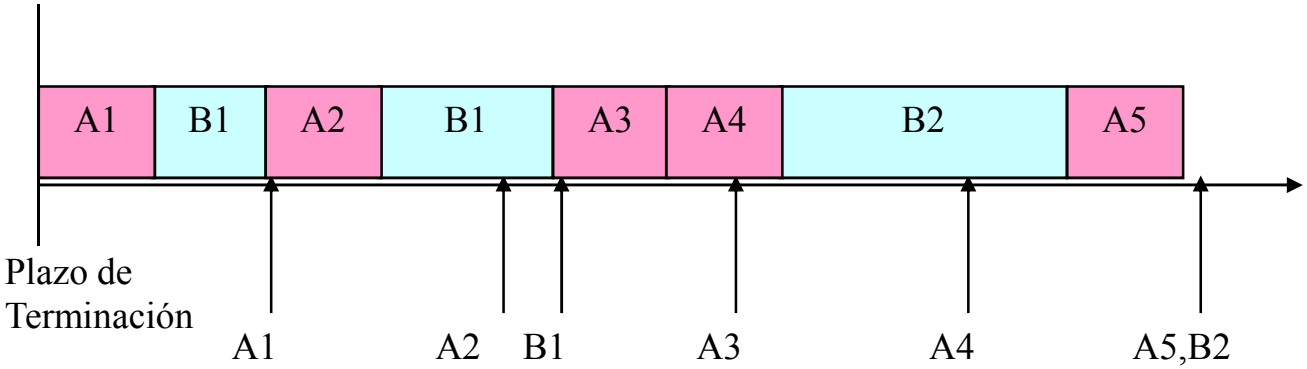
Vemos que en este caso B1 no es ejecutado en su totalidad.

Veamos el mismo caso anterior cuando B tiene mayor prioridad que A



Podemos observar que tanto A1 como A4 no pueden ser ejecutados dentro de sus respectivos plazos de terminación.

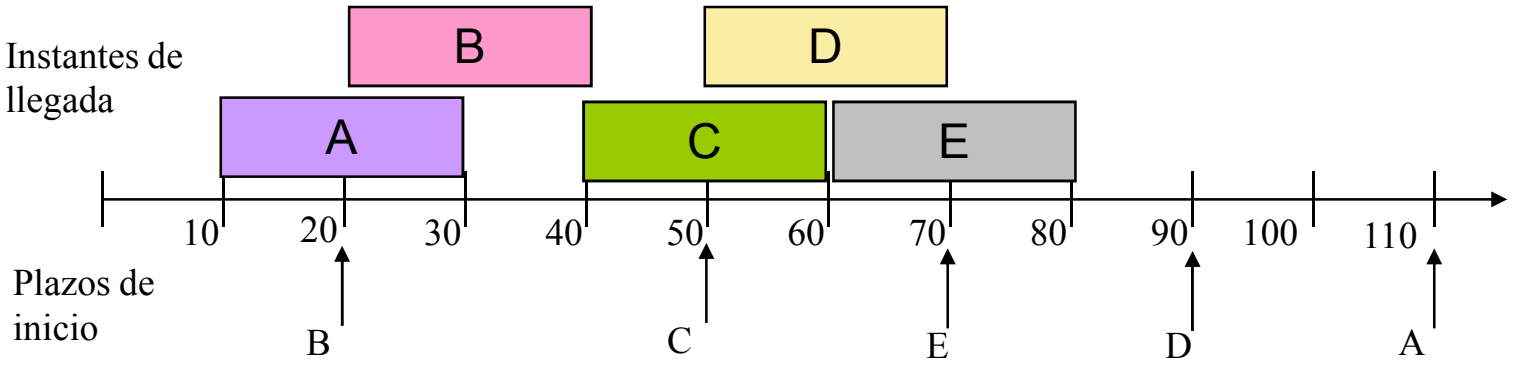
Consideremos ahora el caso de una planificación que, en el instante de apropiación, da prioridad a la tarea con plazo más corto de finalización.



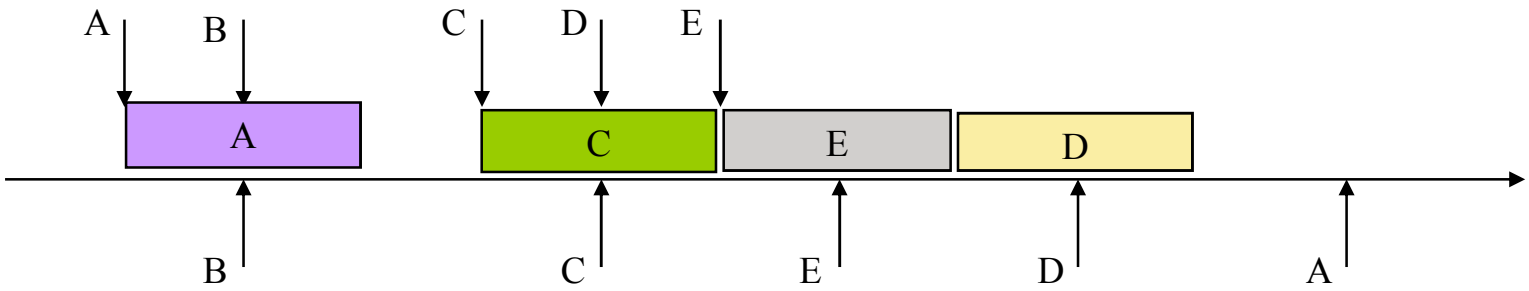
En este caso, pueden cumplirse todos los requisitos del sistema. Puesto que todas las tareas son periódicas y predecibles, se usa un método de planificación con tablas estáticas.

Consideremos un esquema que trate con tareas aperiódicas con plazos de inicio según la siguiente tabla

	Lleg	Ejec.	Plazo
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

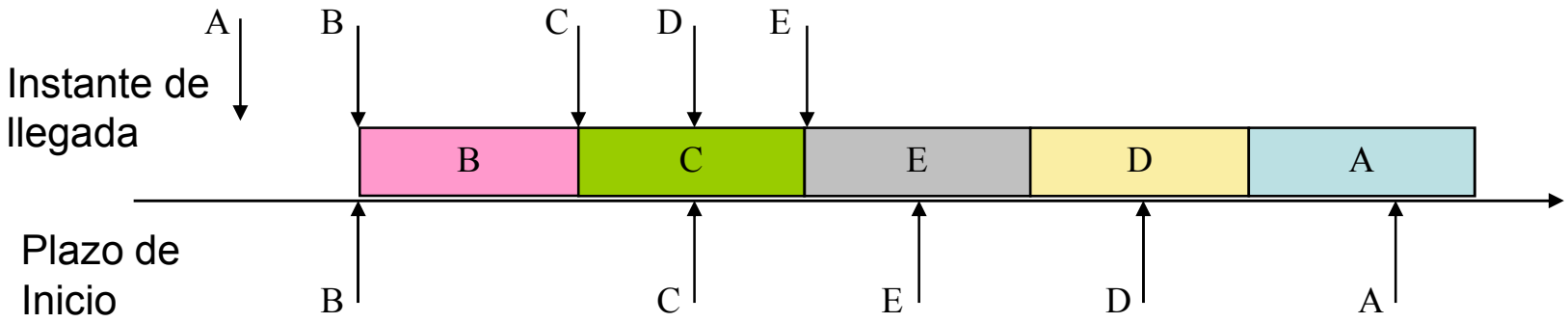


Consideremos en primer lugar un caso de planificación con plazo de inicio



Vemos que en este caso la tarea B no puede ser cumplida, aun cuando requiera servicio inmediato. Este es un riesgo común en la planificación de tareas aperiódicas, especialmente con plazos de inicio.

Una variante de esta política, conocida como la del plazo más próximo con tiempos libres no forzosos, mejora el rendimiento



En este caso, se consideran conocidos los plazos de terminación de cada tarea, antes de que ellas estén listas. Siempre se planifica la tarea con plazo más próximo y se deja ejecutar hasta que finalice.

Métodos dinámicos de planificación:

Se determina la viavilidad durante la ejecución (dinámicamente), en vez de antes de empezar la ejecución (estáticamente). Se acepta una nueva tarea para ejecutar solo si es factible cumplir con sus restricciones de tiempo.

Métodos dinámicos de mejor resultado:

No se realiza ningún análisis de viavilidad, ya que las tareas son generalmente aperiódicas. El sistema intenta cumplir todos los plazos y abandona cualquier proceso ya iniciado y cuyo plazo no se haya cumplido.

Se pueden conseguir mejores resultados realizando la planificación **dinámicamente** en función de los requisitos temporales y de los recursos disponibles

Dos métodos dinámicos interesantes son:

– **Primero el más urgente** (earliest deadline first, EDS)

– **Primero el menos holgado** (least slack first, LSF)

Ambos son óptimos para tareas independientes

– se garantizan los plazos hasta el 100% de utilización

– se comportan bien cuando hay muchas tareas esporádicas

Problemas:

– el comportamiento en caso de sobrecarga es imprevisible

– no está bien resuelta la interacción entre tareas

Fiabilidad y Tolerancia a Fallos

- Fallos de Funcionamiento
  - Los fallos de funcionamiento de un sistema pueden tener su origen en
  - Una especificación inadecuada
  - Errores de diseño del software y/o hardware
  - Averías en el hardware
  - Interferencias transitorias o permanentes en las comunicaciones

Conceptos básicos

La **fiabilidad** (reliability) de un sistema es una medida de su conformidad con una especificación autorizada de su comportamiento

- Una **avería** (failure) es una desviación del comportamiento de un sistema respecto de su especificación
- Las averías se manifiestan en el comportamiento externo del sistema, pero son el resultado de **errores** (errors) internos
- Las causas mecánicas o algorítmicas de los errores se llaman **fallos** (faults)
- Los fallos pueden ser consecuencia de averías en los componentes del sistema

Tipos de fallos

- **Fallos transitorios**
- – desaparecen solos al cabo de un tiempo
- – ejemplo: interferencias en comunicaciones
- **Fallos permanentes**
- – permanecen hasta que se reparan
- – ejemplo: roturas de hardware, errores de software
- **Fallos intermitentes**
- – fallos transitorios que ocurren de vez en cuando
- – ejemplo: calentamiento de un componente de hardware
- Debe impedirse que los fallos de todos estos tipos causen averías

Prevención y tolerancia a fallos

- Hay dos formas de aumentar la fiabilidad de un sistema:

– **Prevención de fallos**

- Se trata de evitar que se introduzcan fallos en el sistema antes de que entre en funcionamiento

– **Tolerancia de fallos**

- Se trata de conseguir que el sistema continúe funcionando aunque se produzcan fallos
- En ambos casos el objetivo es desarrollar sistemas con modos de fallo bien definidos

Prevención de fallos

Se realiza en dos etapas:

- **Evitación de fallos**
  – Se trata de impedir que se introduzcan fallos durante la construcción del sistema
- **Eliminación de fallos**

– Consiste en encontrar y eliminar los fallos que se producen en el sistema una vez construido

Técnicas de evitación de fallos

- **Hardware**
  - Utilización de componentes fiables
  - Técnicas rigurosas de montaje de subsistemas
  - Apantallamiento de hardware
- **Software**
  - Especificación rigurosa o formal de requisitos
  - Métodos de diseño comprobados
  - Lenguajes con abstracción de datos y modularidad
  - Utilización de entornos de desarrollo con computador (CASE) (Computer Aided Software Engineering) adecuados para gestionar los componentes

Técnicas de eliminación de fallos

Comprobaciones

– Revisiones de diseño

– Verificación de programas

– Inspección de hardware (analizadores)

Pruebas (tests)

– Son necesarias, pero tienen problemas:

» no pueden ser nunca exhaustivas

» sólo sirven para mostrar que hay errores, no que no los hay

» a menudo es imposible reproducir las condiciones reales

» los errores de especificación no se detectan

Limitaciones de la prevención de fallos

- Los componentes de hardware fallan, a pesar de las técnicas de prevención
- La prevención es insuficiente si

  – la frecuencia o la duración de las reparaciones es inaceptable

  – no se puede detener el sistema para efectuar operaciones de mantenimiento
- La alternativa es utilizar técnicas de

  tolerancia de fallos

Grados de tolerancia de fallos

- **Tolerancia completa** (fail operational)

  – El sistema sigue funcionando, al menos durante un tiempo, sin perder funcionalidad ni prestaciones
- **Degradación aceptable** (failsoft).

  – El sistema sigue funcionando con una pérdida parcial de funcionalidad o prestaciones hasta la reparación del fallo
- **Parada segura** (failsafe).

  – El sistema se detiene en un estado que asegura la integridad del entorno hasta que se repare el fallo

Redundancia

- La tolerancia de fallos se basa en la **redundancia**
- Se utilizan componentes adicionales para **detectar** los fallos y **recuperar** el comportamiento correcto
- Esto aumenta la complejidad del sistema y puede introducir fallos adicionales
- Es mejor separar los componentes tolerantes del resto del sistema

Tipos de redundancia

- **Redundancia estática**
  – Los componentes redundantes están siempre activos
  – Se utilizan para **enmascarar** los fallos
  – Ejemplo:
    » Redundancia modular triple (ó N)
- **Redundancia dinámica**
  – Los componentes redundantes se activan cuando se detecta un fallo
  – Se basa en la **detección** y posterior **recuperación** de los fallos
  – Ejemplos:
    » sumas de comprobación
    » bits de paridad

Seguridad y fiabilidad

- Un sistema es **seguro** si no se pueden producir situaciones que puedan causar muertes, heridas, enfermedades, ni daños en personas, equipos o medioambiente
- Un **accidente** es un suceso imprevisto que puede producir daños inadmisibles
- Un sistema es **fiable** si cumple sus especificaciones

La seguridad es la probabilidad de que no se produzcan situaciones que puedan conducir a accidentes, independientemente de que se cumpla la especificación o no

Bibliografía

- W. Stallings, Operating Systems, Prentice Hall, 1997.
- Sistemas de Tiempo Real, gsyc-profes@gsyc.escet.urjc.es, 2006.

Confiabilidad

La **confiabilidad** (dependability) es una propiedad de los sistemas que permite confiar justificadamente en el servicio que proporcionan