Sistemas Operativos de tiempo-real

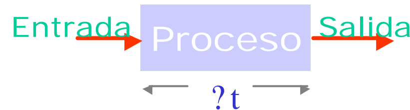
© José Antonio Gómez Hernández, 2001

- Definición de RTS y RTOS.
- Funcionalidad y características.
- Planificación y kernels.
- Estándares.
- Ejemplos: QNX y RT-Linux.

Real-Time for the Real World

Definición

- Un sistema de tiempo-real (RTS) es un sistema de información que procesa entradas asíncronas y produce salidas de un tiempo determinado y acotado.

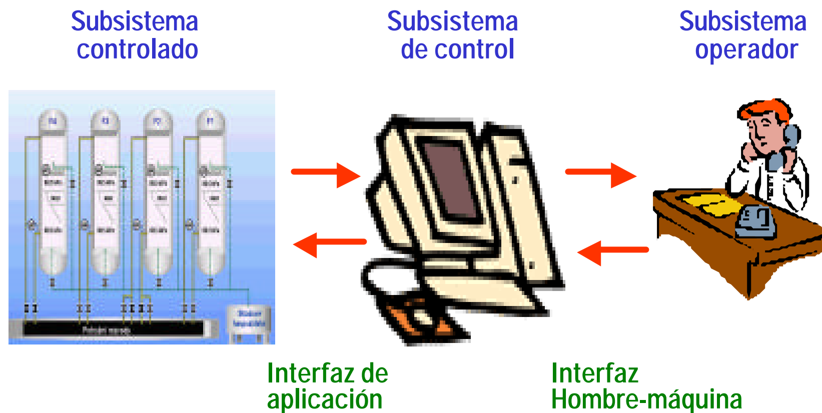


- Debe ser determinista - el tiempo requerido puede calcularse a partir de la carga del sistema, y debe ser acotado.
- Es importante el tiempo de ejecución del peor caso.

La principal responsabilidad de un sistema de tiempo-real (RTS) puede resumirse como aquella que produce resultados correctos mientras satisface unos plazos (*deadline*) predefinidos. Por tanto, la correctitud computacional del sistema depende tanto de la correctitud lógica del resultado que produce como la correctitud temporal, esto es, la habilidad para cumplir los plazos de sus cálculos.

Los sistemas de tiempo-real abarcan un amplio espectro de complejidad desde los más simples microcontroladores (tales como un microcontrolador para un motor de automóvil) hasta sistemas distribuidos (como un sistema de control de tráfico aéreo). Los sistemas futuros serán aun más complejos: control de una estación, espacial, sistemas integrado visión/robótica/AI, conjunto coordinado humano/robots (normalmente en entornos peligrosos como plantas químicas), sistemas de gestión empresarial, sistemas de producción descentralizada, etc.

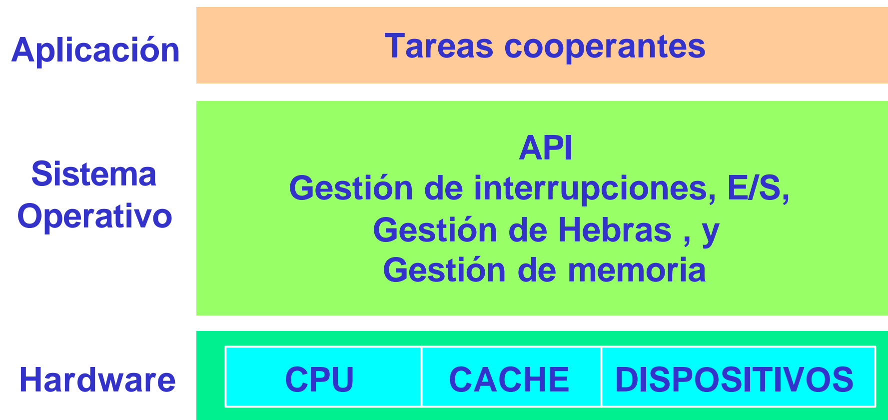
Estructura general de un RTS



Un sistema de tiempo-real genérico puede describirse como un sistema que consta de los siguientes tres componentes. El ***subsistema controlado*** representa la aplicación o entorno (p.ej. una planta industrial), que dicta los requisitos de tiempo real; el ***subsistema de control*** gobierna algunos equipos de computación y comunicación para uso desde el sistema controlado. El ***subsistema operador*** inicia y vigila la actividad del sistema entero. La interfaz entre el sistema controlado y el de control consta de una serie de dispositivos como sensores y actuadores. La interfaz entre el subsistema de control y el operador consiste en una interfaz hombre-máquina.

El subsistema controlado esta implementado mediante tareas, tareas de aplicación, que se ejecutan utilizando los equipos gobernados por el subsistema de control. Este último subsistema puede construirse mediante un número elevado de procesadores, equipados con recursos locales como memoria y discos, e interconectados por una red de área local de tiempo-real. Estos procesadores y recursos estan gobernados por un *sistema operativo de tiempo-real* (RTOS).

El subsistema de control



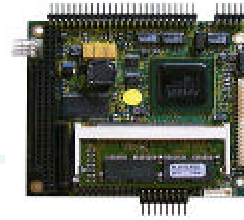
6/11/2001 Diseño de Sistemas Operativos 4

Martin Timmerman define un sistema de tiempo real como aquel que responde de una manera predecible (temporalmente) a la llegada de estímulos externos inpredecibles.

De esta forma, para construir un sistema predecible, todos sus componentes (hardware y software) deben satisfacer estos requisitos. Por ejemplo, el tráfico en un bus debe ocurrir de manera que todos los eventos puedan ser manejados dentro del límite de tiempo prescrito. Un RTOS debe tener todas las características necesarias para ser un buen bloque de construcción para un RTS. Sin embargo, no debemos olvidar que un buen RTOS es sólo un bloque de cosntrucción, y utilizado en un sistema mal diseñado puede producir un sistema que funcione mal.

Probablemente, se pueden construir RTS sin un sistema operativo, pero creemos que un sistema operativo puede ofrecer robustez, transportabilidad, y rendimiento de tiempo-real fiable a una aplicación.

El hardware



- En un RTS, el hardware necesita ser determinista.
- Algunos elementos que no dan respuestas de RT:
  - El uso de pipeline y predicción de saltos en los procesadores actuales introduce latencias impredecibles.
  - El uso de caches añade sobrecarga de ejecución cuando se produce una falta de caché. Esto hace impredecibles los periodos del peor caso para cambios de espacio de direcciones y hebras
  - Dispositivos como tarjetas de red que utilizan Ethernet como protocolo no son aconsejables para construir RTS: El protocolo CSMA/CD es inherentemente no determinista.

El conocimiento del hardware subyacente a un sistema de tiempo-real mejora el uso eficiente de los recursos tanto hardware como software. Auque el papel actual de la teoría de lenguajes es aislar al programador del hardware de base, la construcción práctica de RTS muestra que esto es imposible (si no en la etapa de diseño, ciertamente en las etapas de integración hardware-software.

Para un RTS, el hardware subyacente debe ser determinista. Algunos aspectos del hardware introducen respuestas que no son de tiempo real provocando conductas erróneas en el comportamiento del sistema. La siguiente lista recoge algunos puntos clave que no los únicos y deben considerarse como ejemplos:

•Los procesadores actuales utilizan masivamente mecanismo *pipeline* y de predicción de saltos que introducen impredicibilidad.

•El uso de caché añade sobrecarga a la ejecución cuando se accede a un segmento de código o datos que producen faltas de caché. De esta forma los tiempos del peor caso para conmutación entre hebras y espacios de direcciones en sistemas con caché se hacen impredecibles.

•Ciertos dispositivos físicos como tarjetas de red que utilizan Ethernet como protocolo físico no son aconsejables para la construcción de RTS: el protocolo de acceso al medio CSMA/CD es inherentemente no determinista debido a que se basa en un algoritmo de acceso exponencial aleatorio para resolver las colisiones de red entre múltiples nodos contendientes.

Otros aspectos a tener en cuenta van desde el modo de direccionamiento, la existencia o no de coprocesaror, DMA, etc.

Aplicaciones de tiempo-real

- Suelen modelarse como un conjunto de tareas cooperantes.
- Podemos clasificar estas tareas como:
  - Tareas de tiempo-real duras – una tarea crítica debe completarse en un ?t garantizado para que el resultado no sea catastrófico.
  - Tareas de tiempo-real blandas – son menos restrictivas, y solo requieren recibir más prioridad sobre las menos críticas, ya que el sistema se puede recuperar si no se cumple un plazo.
  - Tareas no de tiempo-real – no tienen plazos.

Una aplicación RT se modela como un conjunto de tareas cooperantes que se clasifican según sus requisitos temporales como:

? **Tareas de tiempo-real duras** - tareas cuya ejecución oportuna (y correctitud lógica) es juzgada como crítica para el funcionamiento del sistema completo. El plazo de tiempo asociado con tareas HRT se denomina *plazo duro* (hard deadline), de acuerdo con la naturaleza crítica de la tarea. Como consecuencia, se supone que el incumplimiento de un plazo duro puede producir un fallo catastrófico del sistema.

? **Tareas de tiempo-real blandas** - estas tareas se caracterizan por plazos de ejecución cuyo cumplimiento es deseable, aunque no crítico, para el funcionamiento del sistema completo, así los plazos asociados a tareas SRT se denominan *plazos blandos*.

? **Tareas no de tiempo-real** - Aquellas que no exiben requisitos de tiempo-real (p. ej. tareas de mantenimiento que puede ejecutarse ocasionalmente de fondo).

El tipo de tarea identifica tanto al RTS como al RTOS. Los ***sistemas de tiempo-real "duros"*** (HRT) son aquellos en los que una falta de aderencia a los citados plazos puede producir un fallo del sistema, mientras que en ***sistemas de tiempo-real "blandos"*** (SRT) el no cumplimiento de plazos no provoca un fallo del sistema.

La complejidad de diseño de los sistemas HRT y SRT esta dominada por cuestiones tales como la temporización de aplicaciones y requisitos de recursos, y la disponibilidad de recursos del sistema. En particular, en sistemas HRT que soportan aplicaciones críticas (p. ej., sistemas de control de vuelo, sistemas de control de plantas nucleares, o sistemas de control ferroviario), la complejidad de diseño puede verse aumentada tanto por la posibilidad de que las aplicaciones tengan requisitos conflictivos como por la demanda de servicios altamente disponible y altamente fiables, bajo cargas del sistema especificadas e hipótesis de fallo, y la necesidad de suministrar esos servicios mientras se satisfacen las fuertes restricciones temporales. En particular, como los HRT deben suministrar servicios oportunos y altamente disponible, el diseño de tales sistemas requiere de las técnicas de tolerancia a fallos apropiadas.

Tareas de aplicación

- Otra clasificación de las tareas de aplicación:
  - Tareas periódicas – se ejecutan a intervalos regulares.
  - Tareas aperiódicas o asíncronas – su tiempo de ejecución no puede anticiparse, si no que esta determinado por la ocurrencia de un evento (interno o externo)
  - Tareas esporádicas – tareas aperiódicas con plazos duros.

6/11/2001 Diseño de Sistemas Operativos 7

Las aplicaciones de tiempo-real pueden además clasificarse como:

? **Periódicas** - Son aquellas tareas que entran en el estado de ejecución a intervalos regulares de tiempo, p. ej. cada T unidades de tiempo. Estas tareas, normalmente utilizadas en aplicaciones tales como procesamiento de señales y control, son típicamente caracterizadas por plazos duros.

? **Aperiódicas** (o asíncronas) - aquellas cuyo tiempo de ejecución no puede ser anticipado, ya que su ejecución esta determinada por la ocurrencia de algún evento externo o interno (p. ej. una tarea que responde a la solicitud de un operador). Estas tareas están caracterizadas normalmente por plazos blandos.

? **Esporádicas** - las tareas aperiódicas caracterizadas por plazos duros se denominan tareas esporádicas (p. ej. tareas relacionadas con la ocurrencia de un fallo del sistema, o con la solicitud de emergencia del operador).

Las aplicaciones de los RTOS han variado con el tiempo. Entre las aplicaciones típicas podemos encontrar los sistemas de control y/o monitorización de procesos, como pueden ser los sistemas de control industrial, sistemas de inyección de combustible en motores de vehículos, sistemas de imágenes medicas, sistemas domóticos, control de armas, etc. Con el desarrollo de los sistemas de tiempo real soft su uso se expande notablemente a áreas como sistemas multimedia, realidad virtual, exploración submarina o planetaria, etc.

El desarrollo de RTOS en entornos de seguridad crítica (p. ej., sistemas de navegación y guiado) impone unos requisitos de seguridad severos a su diseño e implementación. Esto requisitos pueden definirse en términos de probabilidad aceptable máxima de fallo del sistema. Así, por ejemplo, un sistema de control de vuelo requiere una probabilidad de fallo de 10^{-10} por hora de vuelo. Un sistema de control de vehículos en el cual el coste de un fallo puede cuantificarse en términos económicos más que en perdida de vidas humanas (p. ej. sistemas de guía de satélites), requiere una probabilidad de fallo por hora de 10^{-6} a 10^{-7}.

SOs de tiempo-real (RTOS)

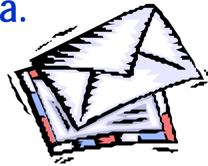
- La característica principal de un RTOS, al igual que en un TRS, es que es determinista, es decir, garantiza que el sistema software se ejecutará dentro de una restricción temporal especificada.
- Podemos clasificarlos como:
  - Kernels propietarios: *QNX, LynxOS, VRTX32, VxWorks, ...*
  - Extensiones de TR de SOs comerciales: *RT-Unix, Solaris,*
  - Kernels de investigación: *MARS, ARTS, CHAOS, ...*

El subsistema controlado esta implementado mediante tareas, tareas de aplicación, que se ejecutan utilizando los equipos gobernados por el subsistema de control. Este último subsistema puede construirse mediante un número elevado de procesadores, equipados con recursos locales como memoria y discos, e interconectados por una red de área local de tiempo-real. Estos procesadores y recursos están gobernados por un *sistema operativo de tiempo-real* (RTOS). De acuerdo con esto y con la clasificación de tareas vista, podíamos pensar que la principal responsabilidad de un RTOS es garantizar que la ejecución individual de cada tarea satisfaga los requisitos temporales impuestos. Sin embargo, con objeto de cumplir esta responsabilidad, el objetivo del RTOS no puede establecerse sólo con minimizar el tiempo de respuesta medio de cada tarea de aplicación; en su lugar, el aspecto fundamental del RTOS es que sea predecible, es decir, la conducta temporal y funcional de un RTOS debe ser tan determinista como sea necesario para satisfacer la especificación. Así, un hardware rápido y unos algoritmos eficientes son útiles al objeto de construir un RTOS; sin embargo, no son suficientes para garantizar la conducta predecible necesaria para tal sistema.

Para que una colección de tareas cooperantes pueda tener lo recursos que necesita, es obvio que un buen RTOS debe ser capaza de realizar una asignación de recursos y una planificación de la CPU integradas. Pero no sólo eso, es también necesario acotar las primitivas que suministra.

Paradigmas de diseño de RTOS

- RTOS activados por eventos - cualquier actividad se iniciada en respuesta a la ocurrencia de un evento particular causado por en el entorno del sistema.



- RTOS activados por tiempo -las actividades del sistema se inician como instantes predefinidos de un tiempo globalmente sincronizado.

6/11/2001 Diseño de Sistemas Operativos 9

En la literatura podemos encontrar dos paradigmas generales de diseño de RTOS. Estos paradigmas han llevado al desarrollo de dos arquitecturas de RTOS diferentes, denominadas ***arquitecturas activadas por eventos*** (ET) y ***arquitecturas activadas por tiempo*** (TT). En esencia, en ET RTOS cualquier actividad del sistema se inicia en respuesta a la ocurrencia de un evento particular, causado por el entorno del sistema. Por el contrario, en TT RTOS las actividades del sistema se inician como instantes predefinidos de tiempo globalmente sincronizado. En ambos casos, la predecibilidad del RTOS se alcanza utilizando estrategias (diferentes) para evaluar, antes de la ejecución de cada tarea de la aplicación, los recursos necesarios para la tarea, y los recursos disponibles para satisfacer esas necesidades. Sin embargo, en arquitectura ET, las necesidades de recursos y la disponibilidad puede variar en tiempo de ejecución, y deben ser juzgadas dinámicamente. Por el contrario, en arquitecturas TT, las necesidades pueden calcularse fuera de línea, basándonos en análisis anteriores a la ejecución de la aplicación específica; si esas necesidades no pueden ser anticipadas, se utiliza la estimación del peor caso.

Los defensores de las arquitecturas TT critican el enfoque de la arquitectura ET, debido a su propia naturaleza, pueden caracterizarse por un número excesivo de posibles conductas que deben ser analizadas cuidadosamente con objeto de establecer su predecibilidad. Por el contrario, los defensores de las arquitecturas ET indican que estas arquitecturas son más flexibles que las TT, y son ideales para una amplia clase de aplicaciones que no permiten la predeterminar sus requisitos de recursos. En particular, argumentan que las TT, debido al enfoque de la estimación del peor caso, son más propensas a desperdiciar recursos con objeto de suministrar una conducta predecible.

Tanto en las arquitectura ET como TT, los recursos necesarios y el juzgar su disponibilidad debe realizarse teniendo en cuenta los requisitos temporales de las aplicaciones. De aquí, las cuestiones de gestión del tiempo, que caracteriza la conducta temporal del sistema, son cruciales en el diseño de un RTS.

Temas básicos

- Los dos puntos cruciales en un RTOS:
  - Algoritmo de planificación – Su responsabilidad es determinar un orden de ejecución de las tareas de TR que sea factible (que satisfaga los requisitos de tiempo y recursos para esas tareas.
  - El kernel del sistema – que suministra las primitivas y características básicas para una sincronización y comunicación eficientes, respuesta garantiza a las interrupciones, sistema de archivos rápido y fiable, y eficiente gestión de memoria, soporte a E/S, ...

Cuando tenemos una serie de actividades con restricciones temporales, como ocurre en RTS, el primer problema que nos viene a la mente es cómo planificar esas actividades para satisfacer sus restricciones temporales.

Claramente, un RTOS debe de poder realizar una planificación integrada de la CPU y una asignación de recursos de forma que colecciones de tareas cooperantes pueden obtener los recursos que necesitan, en el tiempo correcto. Además de los algoritmos de planificación, las predecibilidad requiere primitivas de sistema operativo acotadas.

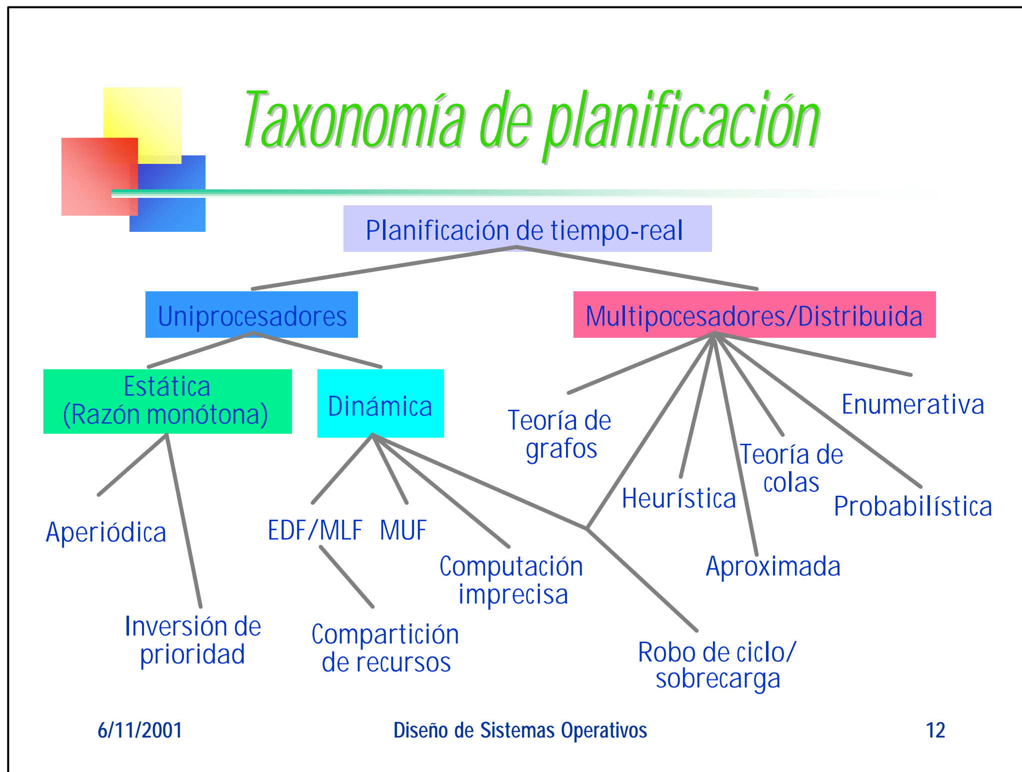
Planificación de tiempo-real

- La elección del algoritmo de planificación depende de muchos factores:
  - # procesadores y características (homegeneos o heterogeneos),
  - relaciones de precedencia entre tareas de la aplicación,
  - método de sincronización,
  - naturaleza de la aplicación (apropiativa o no).

6/11/2001 Diseño de Sistemas Operativos 11

La planificación ha sido uno de los puntos más ampliamente investigados dentro los RTS. Esto se debe a la creencia de que el problema básico en RTS es asegurar la restricciones temporales de las tareas.

Dada la enorme cantidad de literatura del tema, una visión de todos los algoritmos necesitaría un tiempo del que no disponemos por ello solo veremos una clasificación de algoritmos que nos muestre el estado del arte en el tema.



Los enfoques de planificación se pueden dividir toscamente en algoritmos de planificación uniprocesadores y multiprocesadores, donde los enfoques multiprocesadores se utilizan a menudo en planificación de tiempo-real distribuida.

El método estático más comúnmente utilizado son los **planificadores cíclicos**, y más concretamente el de **Razón Monotona** (*RM*), en parte debido a su facilidad de correspondencia con los planificadores basados en prioridades. La idea básica de los algoritmos RM es asignar prioridades fijas y diferentes a tareas con diferentes tasas de ejecución: se asigna mayor prioridad a tareas de mayor frecuencia, y prioridad más baja a las tareas de menor frecuencia. En cualquier instante, el planificador sencillamente elige la tarea con mayor prioridad. Los algoritmos RM presentan básicamente dos problemas: no suministran soporte para el cambio dinámico de periodos de las tareas y/o sus prioridades, y las tareas pueden sufrir **inversión de prioridad**.

Los algoritmos **Earliest Deadline First** (EDF) pueden utilizarse tanto en planificación estática como dinámica. Estos algoritmos utilizan el plazo de una tarea como su prioridad. Como la tarea de menor plazo tiene la mayor prioridad, las prioridades resultantes son naturalmente dinámicas y los periodos de las tareas pueden cambiar en cualquier momento. Una variante de este método, la planificación **Minimum-Laxity-First** (MLF), asigna una laxitud a cada tarea, ejecutando la tarea de menor laxitud. Laxitud mide la cantidad de tiempo restante antes de que el plazo de una tarea termine si la tarea utiliza su tiempo de ejecución máximo asignado. Esencialemente, la laxitud es una medida de la flexibilidad disponible para planificar una tarea.

Mientras que EDF es superior a RM, el problema con EDF es que no hay forma de garantizar que tareas fallaran en situaciones transitorias de sobrecarga. Esto ha desarrollado una variante de EDF, la **Maximum-Urgency-First** (MUF), donde se da a cada tarea una descripción explícita de urgencia. Esta urgencia se define como una combinación de dos prioridades fijas, y una prioridad dinámica, que es inversamente proporcional a la laxitud de la tarea. Las prioridades fijas, una (*criticidad de la tarea*) tiene precedencia sobre la dinámica y la otra (*prioridad de usuario*) tiene menor precedencia. La idea es utilizar las nociones especificadas por el usuario de "prioridad" (esto es, criticidad) para ayudar en

Características de un buen RTOS*

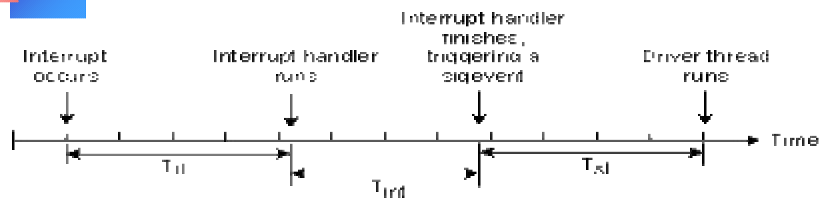
- Debe ser multihebrado y apropiativo.
- Planificación por prioridades ya que no existen SOs controlados por plazos.
- Debe existir mecanismos de sincronización predecibles.
- Debe existir un mecanismo de herencia de prioridad.

(*) www.dedicated-systems.com/encyc/publications/faq/rtfaq.html

Parámetros de fabricación

- De acuerdo con las anteriores característica de fabricante de un RTOS debe especificar claramente los siguientes parámetros:
  - Latencia de interrupción.
  - Tiempo máximo de realización de una llamada.
  - Tiempo máximo que el SO y los manejadores enmascaran las interrupciones.
  - Niveles de interrupción del sistema.
  - Niveles IRQ de los manejadores, el tiempo máximo que toman, ...

Lantencias



T_{il} interrupt latency
T_{int} interrupt processing time
T_{sl} scheduling latency

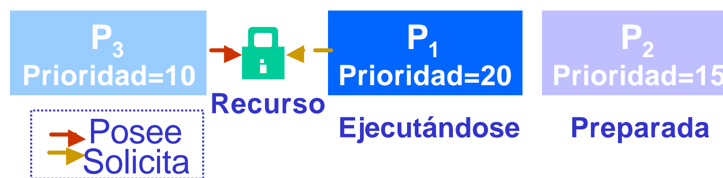
- Por ejemplo, en QNX:
  - Pentium a 166MHz ? T_{il}= 3.3 ?s ; T_{sl}=2.93 ?s
  - 486DX4 a 100MHz ? T_{il}= 5.6 ?s ; T_{sl}=11.1 ?s

Latencia de planificación

- T_{sl} viene determiado por dos factores:
  - Tiempo necesario para apropiar a la hebra en ejecución; de aquí la necesidad de que los kernels sen apropiativos.
  - Al apropiar a la hebra en curso, debemos liberar los recursos que utiliza si estos van a ser usados por la hebra más prioritaria. Si esto no ocurre se dice que se ha producido inversión de prioridad.

Inversión de prioridad

- Fenómeno producido cuando un proceso (o hebra) de cierta prioridad, P_2, se ejecuta antes que otra de mayor prioridad, P_1, debido a que P_1 espera por un recurso que tiene bloqueado otro proceso, P_3.
- Se solventa a través de (i) herencia de prioridad, o (ii) protocolos de tope (*ceiling protocols*).



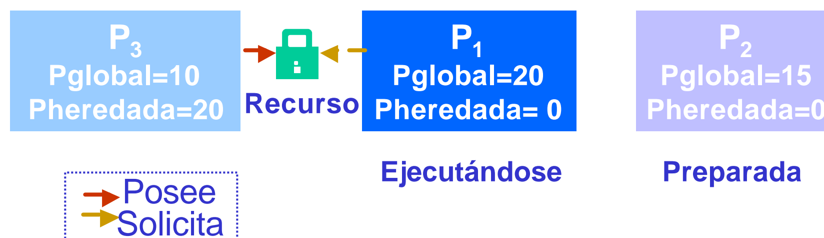
6/11/2001 Diseño de Sistemas Operativos 17

Se produce **inversión de prioridad** cuando una tarea de alta prioridad debe esperar a una de menor prioridad debido a que ciertos recursos necesarios por la tarea de alta prioridad están siendo utilizados por la de baja prioridad (es decir, las tareas esperan sobre secciones críticas).

Se han propuesto diferentes soluciones al problema, como son la **herencia de prioridad**, utilizada por ejemplo en Solaris 2.x, el **protocolo de tope** propuesto por Chen et al.

Herencia de prioridad

- Una hebra tiene dos prioridades: su prioridad global y la heredada. El valor de prioridad para planificar una hebra es el mayor de estos números.
- Cuado una hebra se bloquea en un recurso lega (presta) su prioridad a la hebra poseedora del recurso.
- La propiedad de herencia de prioridad es transitiva.



Ejecutándose

Preparada

Posee
Solicita

La idea básica es que cuando una hebra de prioridad alta se bloquea en un recurso poseído por una hebra de prioridad menor, la hebra de mayor prioridad presta su prioridad a la hebra que posee el recurso, permitiéndole que se ejecute, "acelerando" su ejecución, y por tanto favorecemos que libere más rápidamente los recursos que necesita la hebra de mayor prioridad.

La herencia de prioridad es fácil de implementar, pero presenta el problema de que no es aplicable a todas las primitivas de sincronización. Como se puede ver de su funcionamiento para poder prestar la prioridad debemos ser capaces de conocer que hebra posee el recurso. Esta información debe residir en el cerrojo que protege el recursos. Muchos cerrojo no suministrar esta información (por ejemplo, los semáforos) y por tanto no podemos aplicar el método.

Estándares

- POSIX 1003.1b (antiguo 1003.4): incluye funciones para:
  - E/S síncronas y asíncronas
  - Primitivas IPC
  - Habilidad para bloquear y mantener memoria
  - Planificación por prioridades
  - Archivo de tiempo real, y
  - Cronómetro.
- TROM (The Real-time Operating-system Nucleus). Consta de varios subproyectos entre ellos:
  - ITROM – especificación del RTOS para sistemas empotrados e industriales,
  - ?ITROM - microprocesadores y micro controladores de bajo costo.

6/11/2001 Diseño de Sistemas Operativos 19

Paralelamente al desarrollo y construcción de kernels de sistemas operativo para objetivos específicos de aplicaciones y plataformas arquitectónicas, la industria ha desarrollado estándares para caracterizar la funcionalidad típica ofrecida por el sistema operativo.

Uno de estos esfuerzos es el estándar POSIX.1b (anteriormente 1003.4). Este estándar derivado de Unix, incluye entre otras las siguientes funciones:

- E/S tanto síncronas como asíncronas.
- primitivas IPC (memoria compartida, semáforos, y eventos asíncronos).
- La capacidad para bloquea memoria.
- Planificación por prioridades.
- Archivos de tiempo real
- Cronómetros.

Evaluación de rendimiento

- Existen tres métodos generales:
  - Benchmarks de grano fino o medidas del comportamiento de kernel a bajo nivel, que evalúan el comportamiento hard-soft de las primitivas más frecuentemente utilizadas – *Rhealstone*.
  - Bechmarks orientados a la aplicación que dan una visón de mayor nivel como número de plazos perdidos. *Hartstone*.
  - Evaluación basadas en simulación

•**Bechmarks de grano fino** – muestra los méritos que posee la combinación hardware-software. Un problema es que para medidas muy exactas, uno puede tener acogerse a hardware de propósito especial; de otra forma, el experimento puede ser no repetible.

> •El más conocido es **Rhealstone**. Una "Figura Rhealstone" es la suma ponderada de seis categorías de actividades: tiempo de conmutación entre tareas, tiempo de apropiación de una tarea, tiempo de latencia de interrupción, tiempo de cambio de un semáforo, tiempo de ruptura de un interbloqueo, tiempo de producción de un datagrama.

•**Benchamars orientados a la aplicación** – a menudo se implementan como aplicaciones sintéticas (sus plazos estan obtenidos de ciertas distribuciones) ejecutándose sobre el ejecutivo de tiempo real.

> •El más conocido es **Hartstone**. Consta de tareas periódicas y aperiódicas, y tareas adicionales que representan puntos de sincronización en la aplicación. Las periódicas tiene plazos duros, las aperiódicas, duros y blandos. Las aperiódicas con plazos blandos se ejecutan de fondo, y una planificación optima trata de minimizar su tiempo de retorno (turnaround time) mientras también maximiza el número de plazos alcanzados. Esta prueba es útil para evaluar el sistema como un todo.

•**Simulación** – La ventaja de este método es que se puede modelar el hardware en conjunción con el software, lo que permite entender las propiedades de rendimiento de kernel de tiempo real sobre hardware aún no construido. La desventaja del método es que es difícil contabilizar todas las idiosincrasias de los sistemas actuales.

Ejemplos de RTOS

- Existen multitud de RTOS en el mercado, pero no todos están bien documentados.
- Algunos de los más documentados son:
  - QNX (www.qnx.com)
  - Linux de tiempo-real: (www.realtimelinux.org)
    - RATAI
    - NMT RTLinux
    - Kurt....

- RTOS conforme POSIX1.b que suministra multitarea, planificación apropiativa controlada por prioridades, y un rápido cambio de contexto.
- QNX es muy flexible en cuanto a configuración ya que está construido sobre un microkernel, denominado Neutrino.

6/11/2001 Diseño de Sistemas Operativos 22

QNX consiste en un microkernel y un conjunto de módulos opcionales, que se reparten de manera eficientes las funciones del SO. Cuando se necesita un servicio, por ejemplo, la red o una GUI, simplemente se ejecuta el módulo correspondiente.

QNX ofrece las ventajas de un microkernel: es pequeño, escalable, extensible, y rápido. El microkernel, Neutrino, es altamente fiable, lo suficientemente pequeño para su uso en aplicaciones empotradas en ROM, y lo suficientemente potente para ejecutar una red distribuida de cientos de procesadores.

El sistema de archivo equilibra el tamaño, la funcionalidad y el costo. QNX permite utiliza diferentes sistemas de archivos desde sistemas de archivos POSIX (Fsys), hasta sistemas de archivos CR-ROM, pasando por sistemas de archivos empotrados (Esys), Sistema de archivos SMB (Server Message Block – utilizado por Windows 95 y NT, Samba...), etc.

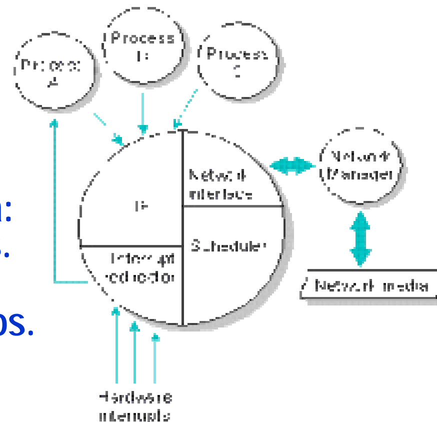
El Gestor de dispositivos posee una alta productividad, y una baja sobrecarga, y suministra la interfaz entre todos los procesos y los dispositivos terminales. Entre sus características podemos citar: escritura totalmente bufereadas, un bufer de entrada por tty que puede ir desde 256 bytes hasta 64 KB para aplicaciones de protocolos de alta velocidad, las solicitudes de entrada pueden retorna por timeout o por datos recibidos, y posee primiticas asíncronas de e/s.

La red de alto rendimiento de QNX (denominada FLEET) es tolerante a fallos, realiza equilibrio de carga "al vuelo", tiene un rendimiento eficiente, un arquitectura eficiente y permite el procesamiento distribuido transparente.

Además, QNX posee un conjunto útil y fácil de usar de herramientas de mantenimiento y diagnóstico, por ejemplo, *alive* muestra el estado de todos los nodos de la red, *nameloc* sondea cada nodo en busca de nombres de procesos, *netboot* acepta solicitudes de arranque de máquinas que intentan arrancar por la red, ...

Estructura de QNX

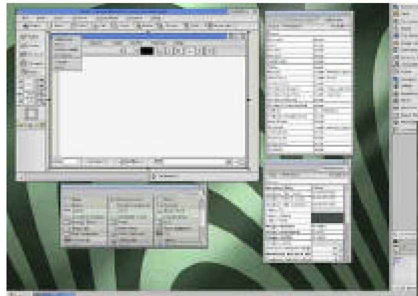
- El ?kernel solo realiza planificación y paso de mensajes.
- Servicios implementandos como procesos de sistema:
  - `Proc`,:Gestor de procesos.
  - `Fsys`: Gestor de archivos.
  - `Dev`: Gestor de dispositivos.
  - `Net`:Gestor de red.



El microkernel maneja la creación de procesos, gestión de memoria, y control de cronómetros. El enfoque para el procesamiento distribuido transparente permite lanzar procesos a través de la red QNX, permitiendo la herencia completa del entorno, incluyendo archivos abiertos, directorio actual, descriptores de archivos, e ID de usuarios. El microkernel también incluye servicios POSIX.1 (certificado) y muchos servicios de tiempo-real POSIX1.b, además de diagnóstico de alta-velocidad de muestreo de eventos.

Entre las principales características podemos citar: (a) relojes y cronómetros POSIX1.b, (b) interrupciones totalmente anidadas, (c) manejadores de interrupciones ligables y eliminables dinámicamente, (d) primitivas de depuración empotradas para depuración local y remota por la red, (e) recursos y límites del sistema configurables por el usuario, (f) capacidades para nombrar procesos a través de la red, y (g) planificación de procesos según el borrador de POSIX.1b.

Interfaz gráfica (Photon)



- Construida como un ?kernel y una serie de procesos cooperantes.

- Bajo requerimiento de memoria .
- Flexible y extensible.
- Conectividad entre plataformas.

6/11/2001 Diseño de Sistemas Operativos 24

Suministra gráficos profesionales de alta calidad incluso en dispositivos empotrados pequeños. Photon da un soporte gráfico escalable, con capacidades multimedia, y una interfaz totalmente personalizable.

Sobre el microkernel se añaden una serie de gestores básicos (entradas, gráficos y gestor de fuentes), y un conjunto de componentes opcionales que cooperan entre sí. Esta arquitectura permite agregar o quitar módulos para escalar desde un pequeño sistema empotrado hasta una potentes estación de trabajo gráfica.

La micro-GUI de Photon permite aplicaciones de internet, electrónica de consumo, navegación en coche, telefonía, instrumentación médica, etc. Permite ver y probar en vivo sus interfaces, depurarlas, y añadirlas al sistema ya que los sistemas de desarrollo y tiempo de ejecución son la misma cosa.

Linux

- Linux esta diseñado para optimizar el rendimiento medio y tratar de repartir imparcialmente la CPU, no considera el peor caso. Por ello no es adecuado en tiempo-real. En concreto:
  - Planificación no predecible (depende de la carga).
  - Resolución baja de cronómetros (10ms).
  - Kernel no apropiativo.
  - Deshabilita interrupciones como mecanismo de sincronización.
  - Utiliza memoria virtual.
  - Reordena solicitudes de E/S por eficiencia.

Linux esta diseñado como un sistema operativo de tiempo compartido, y como tal trata de optimizar los casos medios, pero un sistema de tiempo-real debe tener en cuenta el peor caso. Esto produce contradiciones entes ambos tipos de sistemas, lo que es bueno para el caso medio tiende a deteriorar el peor caso. Un ejemplo clásico es la memoria virtual y la paginación por demanda.

Algunos de los aspectos más relevantes que hacen que Linux no sea un sistema de tiempo-real:

•El planificador esta diseñado para repartir imparcialmente la carga (tiempo compartido) y además no es escalable, es decir, cuanto mayor es carga del sistema peor realiza sus funciones. Esto hace que la planificación no sea predecible.

•La resolución de los cronómetros de Unix es baja para un amplio número de aplicaciones de tiempo-real.

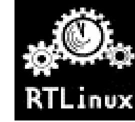
•Como en los Unix tradicionales (no System V ni Solaris), el kernel de Linux utiliza una política de planificación no apropiativa con lo cual es difícil acotar el tiempo de ejecución en este modo.

•Linux utiliza entre otros mecanismos de sincronización la deshabilitación de interrupciones para proteger ciertas secciones criticas, por lo que la atención a los dispositivos se pierde.

•La memorial virtual muy útil en sistemas de propósito general, genera problemas a la hora de acotar el tiempo de ejecución de los procesos.

•Por razones de eficiencia, las colas de peticiones de e/s se reordenan, por ejemplo, para optimizar el acceso a disco, esto introduce impredecibilidad en la e/s de procesos críticos.

Real-time Linux (RT-Linux*)



- Se construye un kernel de tiempo-real bajo Linux. De esta forma, Linux se convierte en una tarea del kernel de tiempo-real que se ejecuta sólo cuando no hay tareas de tiempo-real. Por otra, parte apropiamos Linux, cuando hay tareas de tiempo-real que necesitan la CPU.
- El kernel de tiempo real es no apropiativo, y su retraso máximo de planificación en un Pentium 129 es menor de 20 ?s.

(*) http://luz.cs.nmt.edu/index.html

6/11/2001 Diseño de Sistemas Operativos 26

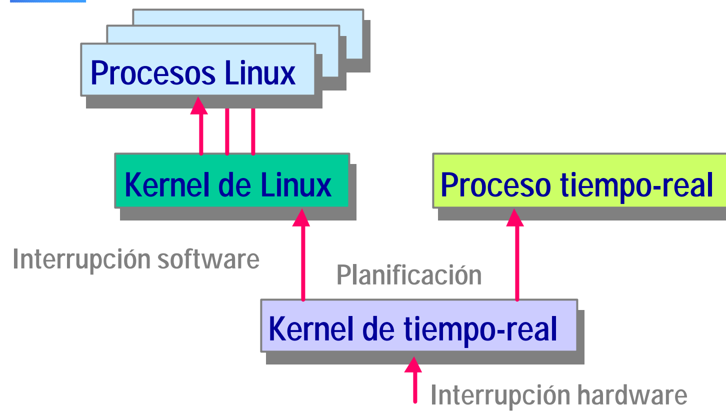
El objetivo de RT-Linux es alcanzar el rendimiento de un RTOS duro, una planificación personalizable con un ajuste temporal fino, y una baja latencia de interrupción, y todo ello con los mínimos cambios del kernel de Linux para poder seguir utilizando todos los servicios de los que dispone el sistemas.

Inspitador en MERT, utiliza el concepto de Máquina Virtual para la emulación de interrupciones, y construye un pequeño RTOS bajo el kernel de Linux. Para este SO, el propio Linux no es más que una tarea nula. Se interpone una capa de emulación sodftware entre el kernel de Linux y el controlador hardware de interrupciones. Esto evita que Linux pueda deshabilitar las interrupciones. Intrupciones como *cli*, *sti* e *iret* son sustituidas por sus versiones software emuladas.

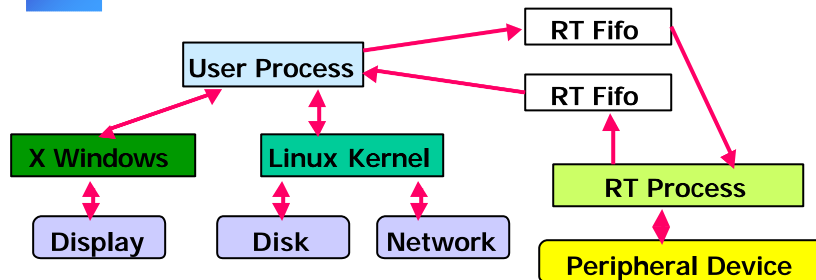
Para evitar la sobrecarga del cambio de contexto, invalidación de TLB, llamadas al sistema, etc., todas las tareas RT se ejecutan en el mismo espacio de direcciones, el del kernel. Se ejcutan como módulos kernel, lo que permite su carga dinámica. La asignación de recursos es estática, es decir, en tareas de RT no se deben utilizar funciones tales como kmalloc.

El propio planificador es un módulo cargable en el kernel. Por defecto, implementa un algoritmo apropiativo basado en prioridades estáticas. Pero también implementa los algorimos RM, y EDF.

Arquitectura de RT-Linux



Estructura de una aplicación



- La comunicación entre un proceso Linux y uno de tiempo real se hace a través de dos FIFOS de tiempo-real (una tarea no se bloquea al leer/escribir en ellas y no se paginan).

Una aplicación de RT-Linux se divide en dos partes: una de tiempo-real que debe ser rápida, simple y pequeña, otra que se ejecuta en espacio de usuario y debe controlar aspectos como e/s, etc. Estas dos tareas se comunican mendiante dos FIFOS de tiempo real (son unidireccionales) – ver figura de la transparencia.

Un FIFO de tiempo real es aquel que se comporta de manera no bloqueante en el lado de la aplicación de RT y de manera convencional en la tarea de usuario. El búfer de los FIFOs se asigna en el espacio de direcciones del kernel.