7919 | Sistemas Embebidos

2º Cuatrimestre de 2017

CLASE 12: TIEMPO REAL Y SISTEMAS OPERATIVOS EMBEBIDOS

Prof: José H. Moyano

Autor original: Sebastián Escarza

Dpto. de Cs. e Ing. de la Computación
Universidad Nacional del Sur
Bahía Blanca, Buenos Aires, Argentina

GrIDSE

Introducción

- Conforme la complejidad del embebido crece, resulta necesaria la inclusión de componentes que resuelvan aspectos usuales:
  - Coordinación de múltiples tareas.
  - Cumplimiento de las restricciones temporales impuestas por la aplicación.
  - Abstracción de detalles de hardware y/o de interfaces de bajo nivel.
  - Implementación de funcionalidad compartida entre múltiples aplicaciones.

Sistemas Operativos Embebidos

Sistemas Operativos Embebidos

- Surgieron para abstraer a las aplicaciones de los recursos de hardware (mayor portabilidad).
- Útiles en aplicaciones de mediana a gran escala con una mayor cantidad de tareas a planificar.
- Van desde pequeños kernels a sistemas completos.
- Similitudes con Sistemas Operativos de escritorio:
  - incluyen alguna forma de multitarea
  - administran recursos de software y hardware
  - proveen servicios a las aplicaciones
  - abstraen del hardware subyacente a las aplicaciones

Sistemas Operativos Embebidos

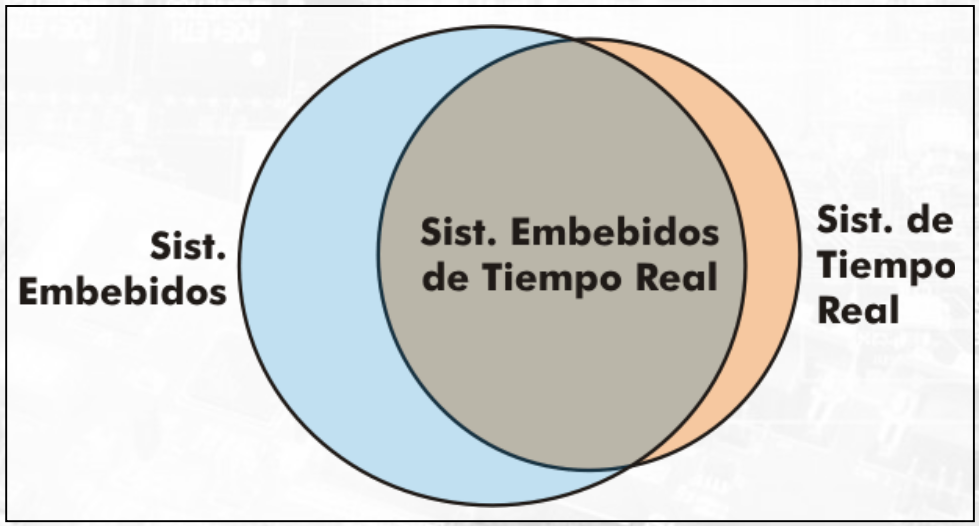
- Diferencias con Sistemas Operativos de escritorio:
  - mayor confiabilidad en ambientes embebidos
  - escalan para alcanzar los requerimientos de la aplicación (sólo se incluyen los servicios necesarios)
  - mayor performance en escenarios limitados
  - requerimientos de memoria reducidos
  - planificación de tareas ajustada a necesidades de tiempo real (se incorpora la noción de correctitud temporal a la correctitud funcional)
  - soporte para sistemas embebidos sin disco que permite bootear ejecutables desde memoria ROM y RAM
  - mayor portabilidad a diferentes plataformas y arquitecturas de hardware

Sistemas Operativos Embebidos

- Diferencias con Sistemas Operativos de escritorio:
  - los sistemas operativos embebidos de menor escala pueden pensarse como librerías:
    - usualmente, en este contexto la aplicación se linkea al sistema operativo
    - no se protegen a si mismos ni a unas tareas de otras como los SO de escritorio
  - los sistemas operativos embebidos de mayor escala constituyen software separado de las aplicaciones, e incorporan facilidades para instalar aplicaciones.
    - mayor similitud con los sistemas de escritorio
    - se incrementan los niveles de protección y la competición por los recursos

Sistemas Operativos Embebidos

- No todos los sistemas operativos de tiempo real son embebidos ni todos los sistemas operativos embebidos son de tiempo real.



- A continuación nos centraremos en RTOS.

Nociones de Tiempo Real

Nociones de Tiempo Real

- Tiempo Compartido vs Tiempo Real.

	Sistemas de Tiempo Compartido	Sistemas de Tiempo Real
Capacidad	Throughput elevado.	Planificabilidad.
Respuesta	Respuesta promedio rápida.	Latencia para el peor caso asegurada.
Sobrecarga	Equidad.	Estabilidad.

  - Planificabilidad es la habilidad de cumplir con todos los deadlines rígidos de las tareas.
  - Estabilidad ante sobrecarga implica que el sistema cumple los deadlines de las tareas críticas, aún en el caso en que no se puedan cumplir todos los deadlines.

Nociones de Tiempo Real

- Planificación de tiempo compartido:
  - Objetivo: maximizar el uso de CPU. Las tareas podrían demorar su conclusión pero el CPU se mantiene continuamente ocupado.

- Ejemplos:
  - FIFO
  - Shortest Job First
  - Priority-based
  - Round-robin (quantum + apropiación)
  - etc.

Nociones de Tiempo Real

- Planificación de tiempo real:
  - Objetivo: asegurar que las tareas críticas cumplen sus deadlines. Se podría sacrificar tiempo de ocupación del CPU para lograr este objetivo.
  - Para las demás tareas no se aseguran tiempos de respuesta (restricciones más blandas de tiempo).
  - Lo usual es que ciertas tareas no críticas o de baja prioridad, utilicen los tiempos "ociosos" de CPU.
  - Análisis del peor caso:
    - tasas/períodos de eventos para tareas más frecuentes posibles
    - tiempo de procesamiento (tareas, ISRs) más elevado posible

Nociones de Tiempo Real

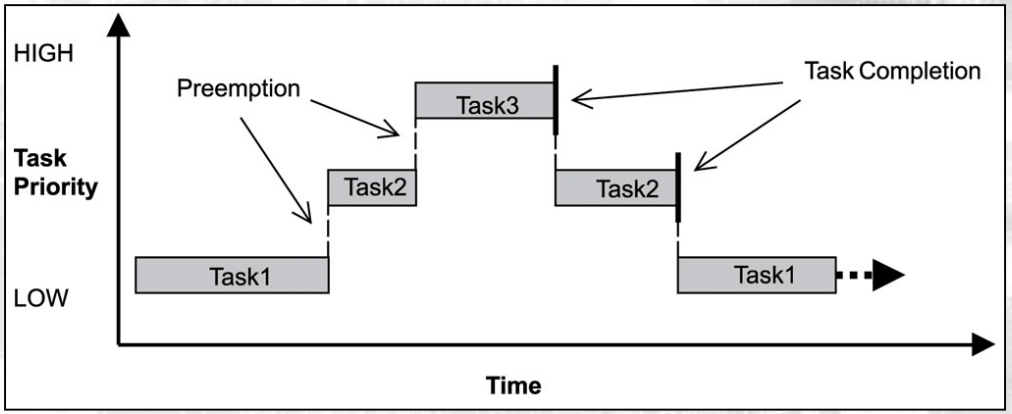
- Planificación de tiempo real - Ejemplos:
  - Real-time cyclic executive: time-slice fijo y deadlines conocidos para cada tarea.
  - Earliest deadline first: ajusta prioridades en función de la tarea con vencimiento más cercano.
  - Minimal laxity first: considera el deadline y el tiempo que falta para completar cada tarea. Se selecciona la tarea que tiene menor tiempo entre el final de su procesamiento y su deadline (laxity mínima o menor margen de planificación).

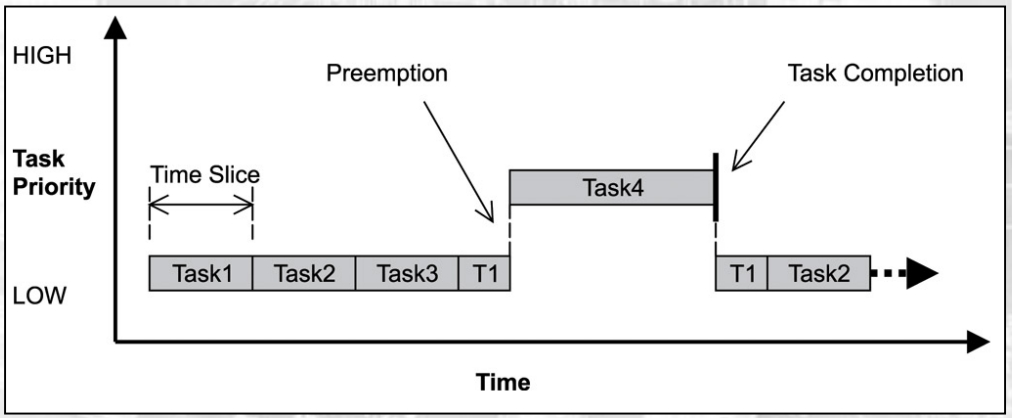
Nociones de Tiempo Real

- Planificación de tiempo real - Ejemplos:
  - Resource reservation: Alternativa para administrar recursos. Tiene implicancias en la planificación al admitir nuevas tareas en función de los recursos que reservan y en cómo la reservación impacta en el cumplimiento de los deadlines establecidos.
  - Rate monotonic scheduling: esquema de asignación de prioridades estático que garantiza la planificabilidad para conjuntos fijos de tareas. Es optimal (si un conjunto de tareas no puede planificarse bajo RMS, no puede ser planificado bajo ningún otro esquema estático).
  - Deadline monotonic scheduling: variante del rate monotonic (p/tareas con deadlines menores a su período).

Nociones de Tiempo Real

- Lo más usual en los RTOS es hallar:
  - planificación basada en prioridades
  - round-robin para tareas con igual prioridad
  - las prioridades de cada tarea se fijan en tiempo de diseño (esquema estático) o dinámicamente (más complejo).





Nociones de Tiempo Real

- En sist. de tiempo real, para cada tarea se considera:
  - Tiempo de apropiación: tiempo que la tarea espera que las tareas de mayor prioridad se completen.
  - Tiempo de ejecución: el tiempo que la tarea demora en realizar su procesamiento.
  - Tiempo de bloqueo: el tiempo que la tarea se ve demorada a la espera de recursos en poder de tareas de menor prioridad.
- Una tarea es planificable si la suma de los tiempos anteriores es menor a su deadline.
- Se recomienda mantener dichos tiempos lo más reducidos que se pueda.

Nociones de Tiempo Real – RMA

- Rate Monotonic Analysis (RMA)
  - Técnica para analizar la planificabilidad de tareas sujetas a deadlines. Es un esquema optimal y estático.
    - 1º: Cumplir las restricciones temporales
    - 2º: Optimizar el uso del procesador
  - Trasciende la planificación (permite analizar temporal-mente aún aquellos sistemas que no utilicen RMS).
- Rate Monotonic Schedulling (RMS)
  - RMS es el algoritmo de planificación que se utiliza para asignar prioridades a las tareas.
  - A tareas más frecuentes (menor período de ejecución), asigna mayor prioridad.

Nociones de Tiempo Real – RMA

- Se asume (en el RMA básico):
  - tareas periódicas
  - tareas independientes (sin interacción)
  - el deadline de una tarea define el comienzo de su próximo período
  - el tiempo de ejecución de cada tarea no varía con el tiempo (esquema estático)
  - todas las tareas tienen el mismo nivel de criticidad
  - las tareas no periódicas se limitan a inicialización y recuperación ante fallos y no tienen deadlines duros

Nociones de Tiempo Real – RMA

- Testeo de planificabilidad (RMA básico):

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq U(n) = n(2^{1/n} - 1) \qquad (1 \leq i \leq n)$$

  - C_i: Tiempo de ejec. en el peor caso de la tarea i
  - T_i: Período de la tarea i
  - U(n): Cota al factor de utilización del CPU (converge al 69% cuando n crece indefinidamente)
  - n: cantidad de tareas a planificar
- Si se cumple la inecuación, existe una planificación que permite a todas las tareas cumplir con sus deadlines.

Nociones de Tiempo Real – RMA

- Ejemplo (RMA básico):

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq U(n) = n(2^{1/n} - 1) \qquad (1 \leq i \leq n)$$

Periodic Task	Execution Time	Period (milliseconds)
Task 1	20	100
Task 2	30	150
Task 3	50	300

$$\frac{20}{100} + \frac{30}{150} + \frac{50}{300} \leq U(3) = 3(2^{1/3} - 1)$$

$$56.67\% \leq U(3) = 77.98\%$$

El conjunto de tareas es planificable!

Nociones de Tiempo Real – RMA

- Extensiones al RMA básico (RMA con bloqueos):
  - Una desigualdad por tarea
  - C_k: Tiempo de ejec. en el peor caso de la tarea k
  - T_k: Período de la tarea k
  - n: cantidad de tareas a planificar
  - B_k: la máxima duración de los bloqueos que puede sufrir la tarea k

$$\frac{C_1}{T_1} + \frac{B_1}{T_1} \leq 1(2^{1/1} - 1) \quad and$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{B_2}{T_2} \leq 2(2^{1/2} - 1) \quad and$$

$$\cdots$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_k}{T_k} + \frac{B_k}{T_k} + \leq k(2^{1/k} - 1) \quad and$$

$$\cdots$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} + \leq n(2^{1/n} - 1)$$

- Se añaden los tiempos de bloqueo (en el peor caso) al modelo básico y se consideran las prioridades.

Nociones de Tiempo Real – RMA

- Ejemplo (RMA con bloqueos):

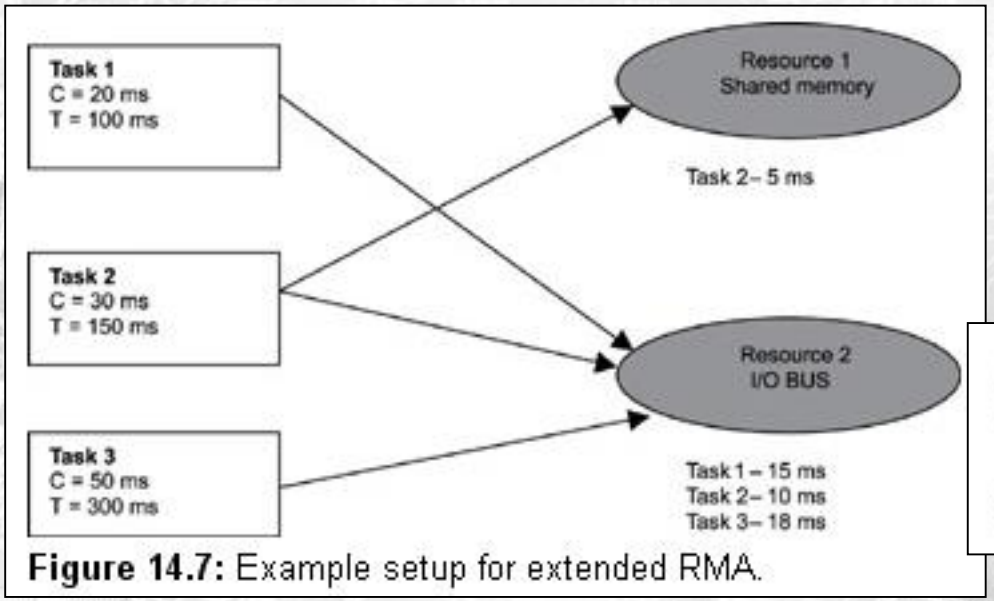


Figure 14.7: Example setup for extended RMA.

$$\frac{20}{100} + \frac{18}{100} \leq U(1) = 1(2-1)$$

$$38\% \leq U(1) = 100\%$$

$$\frac{20}{100} + \frac{30}{150} + \frac{18}{150} \leq U(2) = 2(2^{1/2} - 1)$$

$$52\% \leq U(2) = 82.84\%$$

$$\frac{20}{100} + \frac{30}{150} + \frac{50}{300} \leq U(3) = 3(2^{1/3} - 1)$$

$$56.67\% \leq U(3) = 77.98\%$$

El conjunto de tareas también es planificable!

Nociones de Tiempo Real – RMA

- Además de la extensión para contemplar sincronización entre tareas, existen otras extensiones al Testeo RMA básico que modelan:
  - tareas aperiódicas
  - tareas con tiempos de ejecución variable
  - inversión de prioridad
  - etc.

Tiempo Real – Problemas

- Los principales problemas en presencia de sincronización entre tareas y recursos compartidos son:
  - Deadlocks
  - Inversión de prioridad
- Desde una perspectiva de tiempo real, dichos problemas pueden impedir que el sistema asegure el cumplimiento de los deadlines.
- Además de los problemas funcionales usuales, ocasionan problemas en el temporizado del sistema.

Realtime y Deadlocks

- Un deadlock ocurre cuando múltiples hilos de ejecución se bloquean permanentemente, a raíz de requerimientos de recursos que no pueden satisfacerse.

- Condiciones:
  - Exclusión mutua (acceso exclusivo a recursos).
  - No apropiación de recursos.
  - Retener los recursos adquiridos mientras se espera por los solicitados.
  - Espera circular.

Realtime y Deadlocks

- Ejemplo:

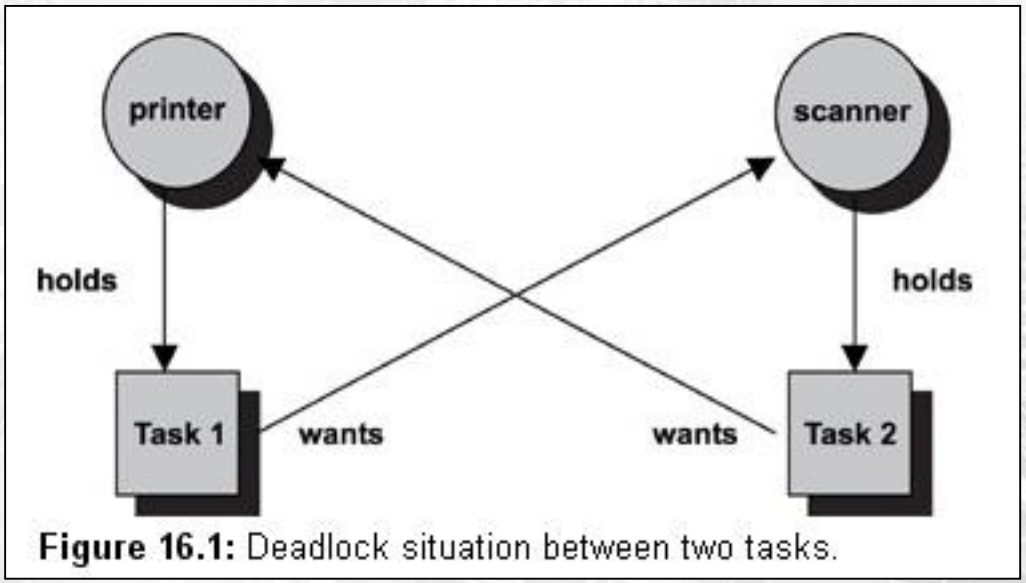


Figure 16.1: Deadlock situation between two tasks.

Realtime e Inversión de Prioridad

- La inversión de prioridad se da cuando una tarea de baja prioridad logra ejecutarse mientras una taea de mayor prioridad espera por recursos.
  - Ejemplo: Inversión de prioridad acotada.

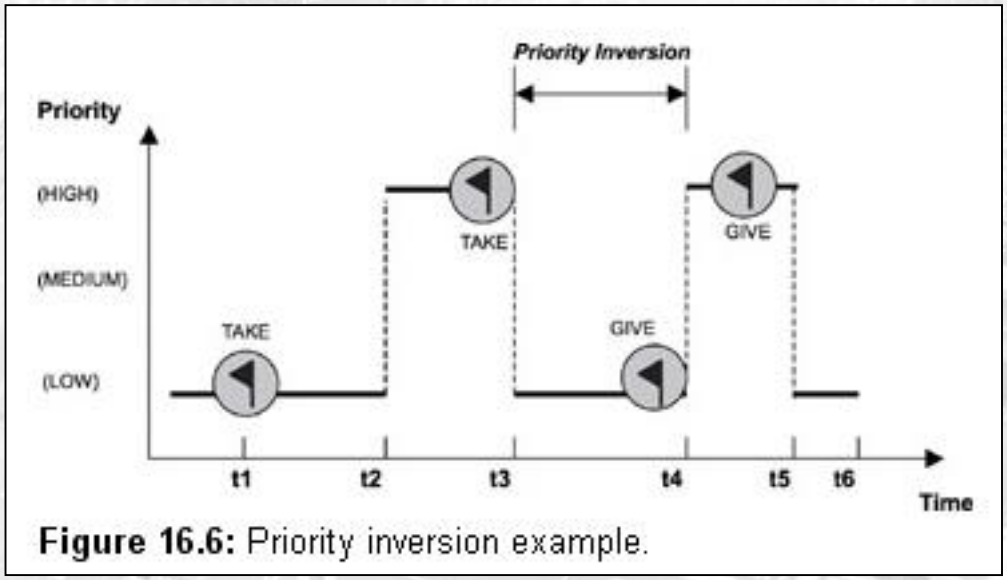
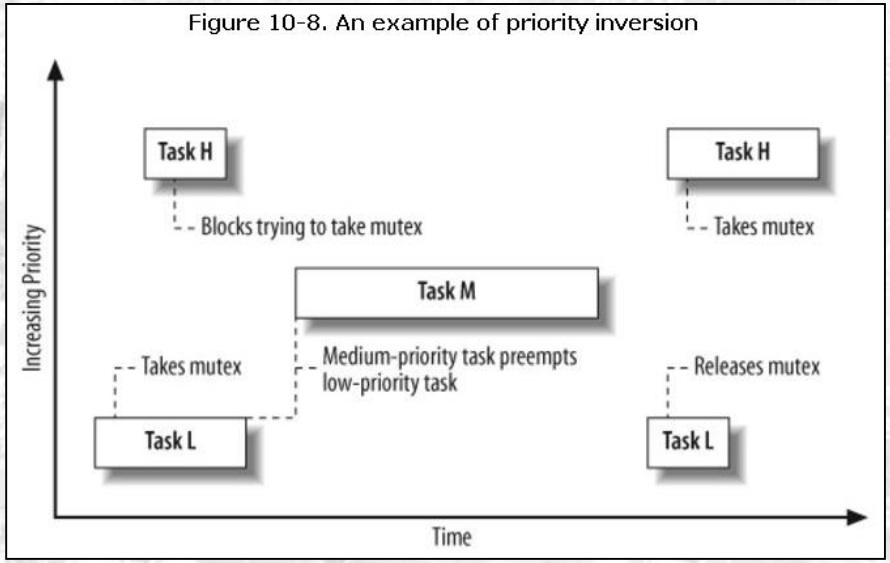


Figure 16.6: Priority inversion example.

Realtime e Inversión de Prioridad

- Ejemplo: Inversión de prioridad no acotada. La tarea H puede demorarse por una tarea no relacionada (M).



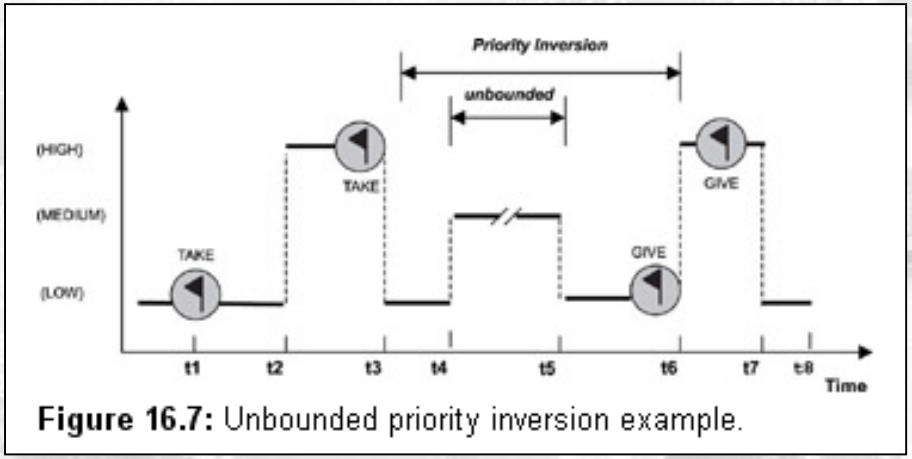


Figure 16.7: Unbounded priority inversion example.

Realtime e Inversión de Prioridad

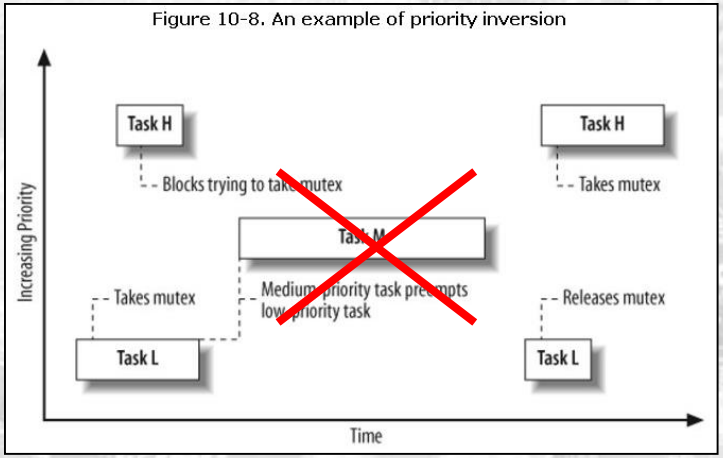
- Problemas de la Inversión de Prioridad no acotada:
  - Se invierten las prioridades. Una tarea de menor prioridad (M) puede ejecutar ante una de mayor prioridad (H).
  - La demora en la atención de una tarea de alta prioridad (H) puede depender de virtualmente cualquier tarea del sistema (M) no relacionada.
  - No se puede acotar el conjunto de tareas a analizar.
  - No se puede acotar el tiempo en que la tarea de alta prioridad es demorada. M es una tarea arbitraria que ni siquiera bloqueó el recurso.
  - El escenario empeora ya que las tareas de mayor prioridad tienen deadlines más cortos (pérdida de deadlines).

Realtime e Inversión de Prioridad

- La inversión de prioridades no puede evitarse, pero si puede mitigarse mediante un protocolo de control de acceso a recursos:
  - Priority Inheritance Protocol (PIP)
  - Ceiling Priority Protocol (CPP)
  - Priority Ceiling Protocol (PCP)

Realtime e Inversión de Prioridad

- Priority Inheritance Protocol (PIP):
  - Las tareas de baja prioridad (L) heredan la prioridad de toda tarea de alta prioridad (H) que espera por un recurso que estas han bloqueado.
  - El RTOS modifica la prioridad de L mientras se comparte el recurso. Se impide que una tarea de prioridad media (M) acceda al CPU.



Realtime e Inversión de Prioridad

- Priority Inheritance Protocol (PIP):
  - El temporizado de H vuelve a quedar acotado únicamente por su tiempo de procesamiento, por el tiempo de apropiación de tarea de mayor prioridad y por el tiempo que L bloquea el recurso.
  - Promoción transitiva de prioridades.

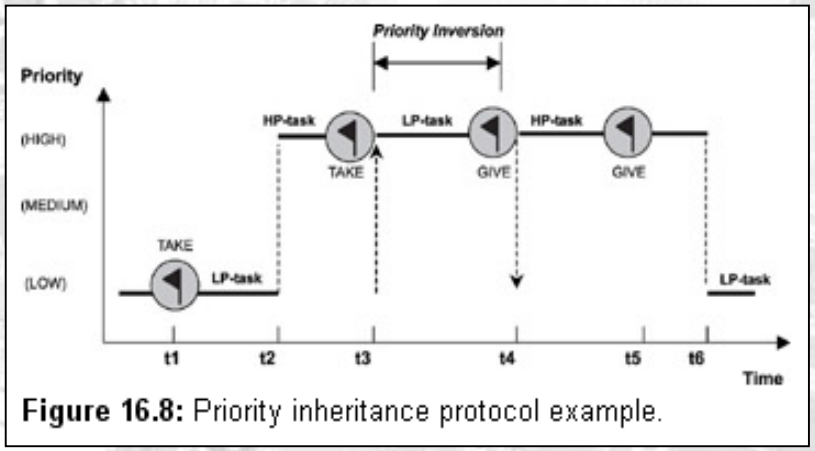


Figure 16.8: Priority inheritance protocol example.

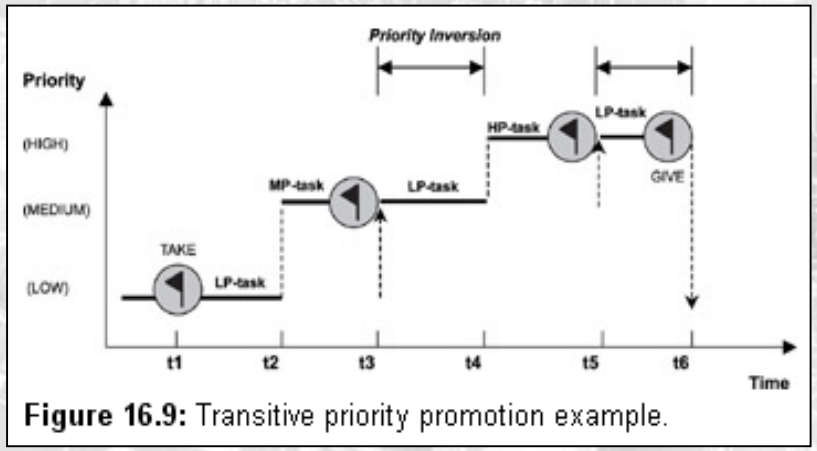


Figure 16.9: Transitive priority promotion example.

Realtime e Inversión de Prioridad

- Ceiling Priority Protocol (CPP):
  - Se conocen por anticipado las prioridades de las tareas y los recursos que estas necesitan.
  - Para un recurso R, la Ceiling Priority es la mayor de las prioridades de todas las tareas que podrían utilizar R.
  - Se incrementa la prioridad a la Ceiling Priority de R.

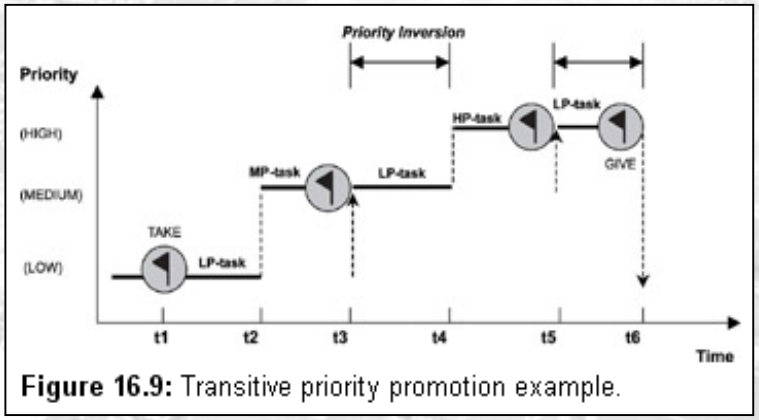


Figure 16.9: Transitive priority promotion example.

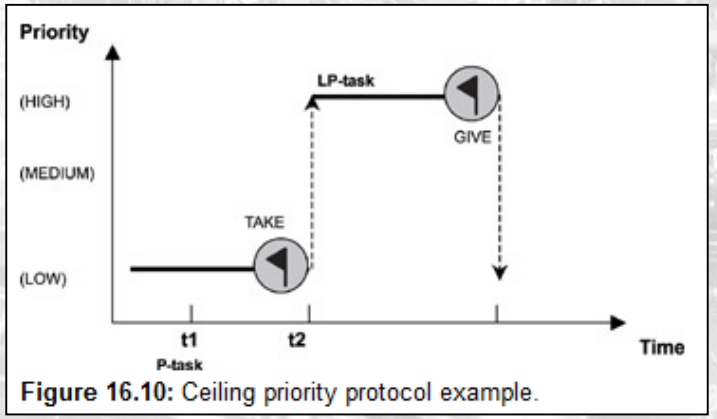


Figure 16.10: Ceiling priority protocol example.

Realtime e Inversión de Prioridad

- Ceiling Priority Protocol (CPP):
  - Cada tarea hereda la prioridad del recurso R aún cuando no haya tareas de mayor prioridad compitiendo por R.
  - Se puede impedir la ejecución de tareas de prioridad media que no compiten por el recurso.
  - Todas las tareas que comparten un recurso R, compiten en simultáneo por ejecutar sus secciones críticas para acceder a R. Se diluyen las prioridades entre las diversas tareas ya que al competir por R son tratadas como si tuvieran la misma prioridad. Conviene reducir al mínimo las secciones críticas para minimizar los conflictos.

Realtime e Inversión de Prioridad

- Priority Ceiling Protocol (PCP):
  - Se conocen las prioridades y qué recursos usan las tareas.
  - Current Priority Ceiling (CPC): La Ceiling Priority más alta de todos los recursos en uso en un instante de tiempo.
- Si la tarea T solicitua un recurso R:
  - Se bloquea si el recurso está reservado.
  - Se otorga si la prioridad de T es mayor que la CPC o la CPC corresponde a algún recurso controlado por T.
  - Se bloquea si la CPC es mayor a todos los recursos controlados por T
- Una tarea T que bloquea T' hereda su prioridad hasta liberar todos los recursos con ceiling priority mayor o igual que la prioridad de T'.

Realtime e Inversión de Prioridad

- Bajo CPC:
  - Una tarea que solicita un recurso sólo puede ser bloqueada por otra tarea y el tiempo de bloqueo se corresponde con la sección crítica de esa otra tarea.
  - Los recursos quedan ordenados por sus Ceiling Priorities y son adquiridos siguiendo dicho orden (prevención de deadlocks).
  - No ocurren bloqueos transitivos.

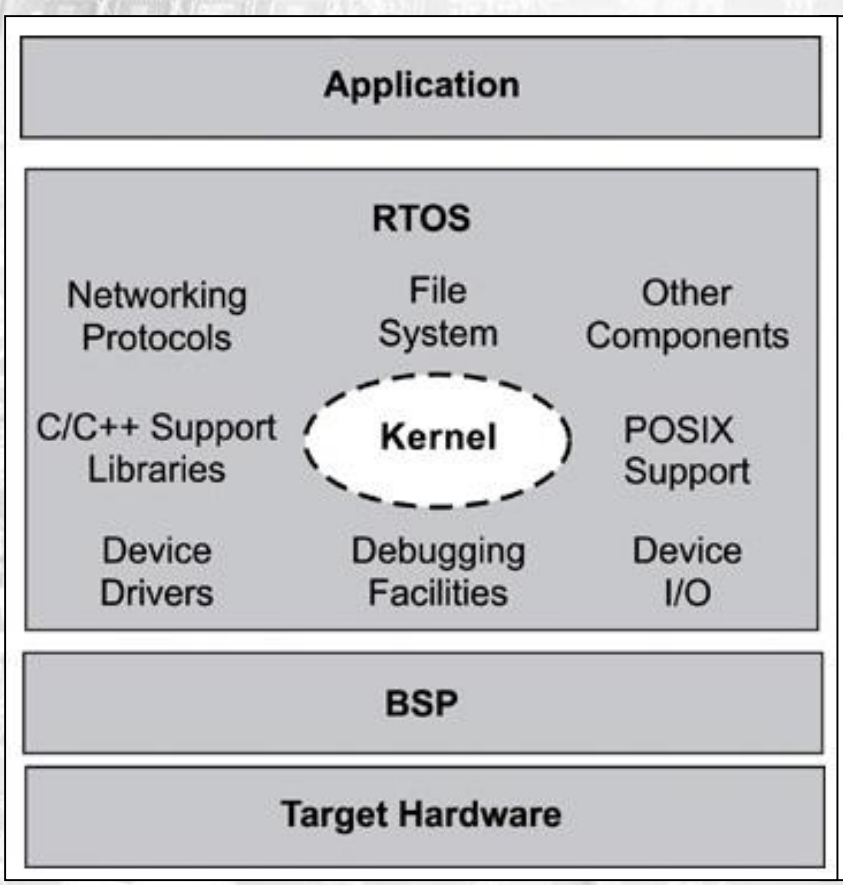
Sistemas Operativos de Tiempo Real (RTOS)

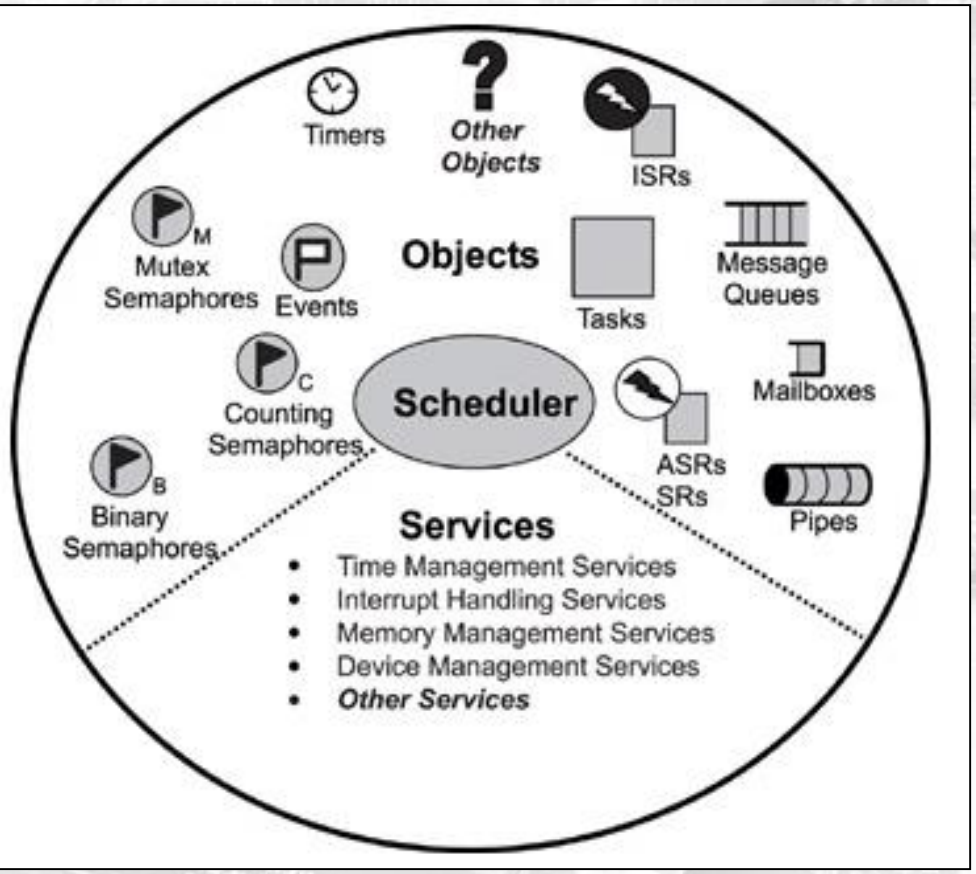
RTOS – Definición

- Definición: Un sistema operativo de tiempo real es un programa que:
  - planifica la ejecución de tareas teniendo en cuenta restricciones de tiempo
  - administra los recursos del sistema
  - provee una base consistente sobre la cual desarrollar las aplicaciones

- De mínima, el kernel (microkernel, monolítico, etc.)
- Adicionalmente se pueden añadir módulos que implementen los servicios requeridos.

RTOS – Principales elementos





RTOS – Principales elementos

- Distintos modelos:
  - modelo por capas
  - microkernel

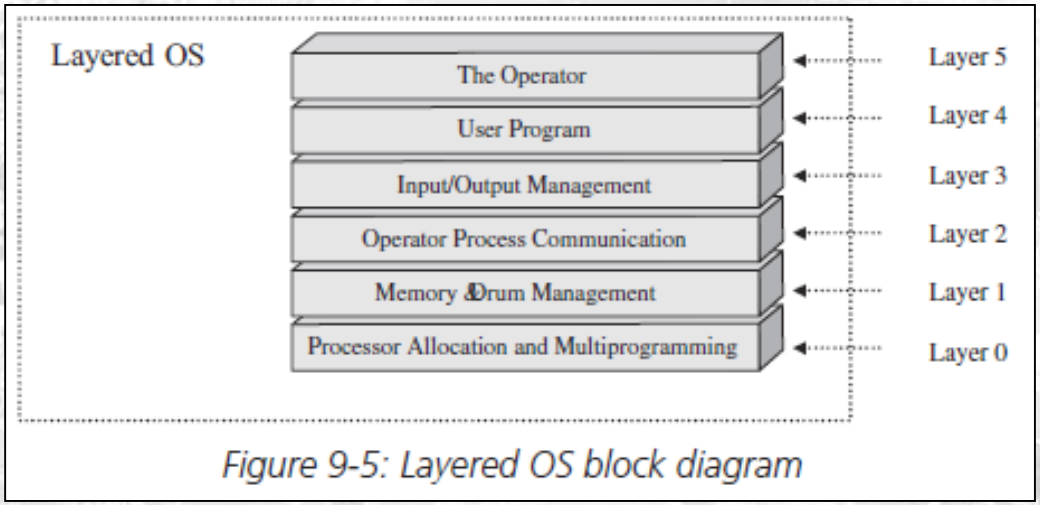


Figure 9-5: Layered OS block diagram

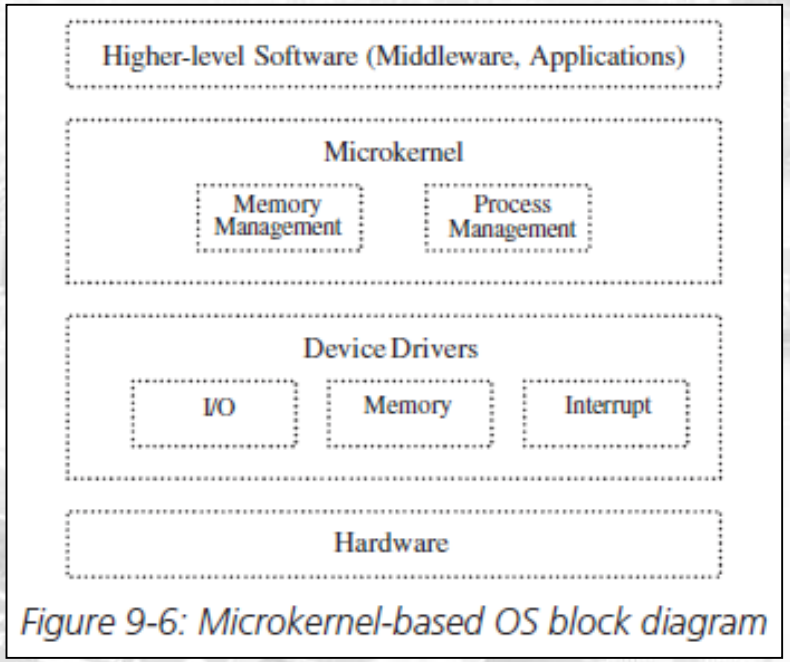


Figure 9-6: Microkernel-based OS block diagram

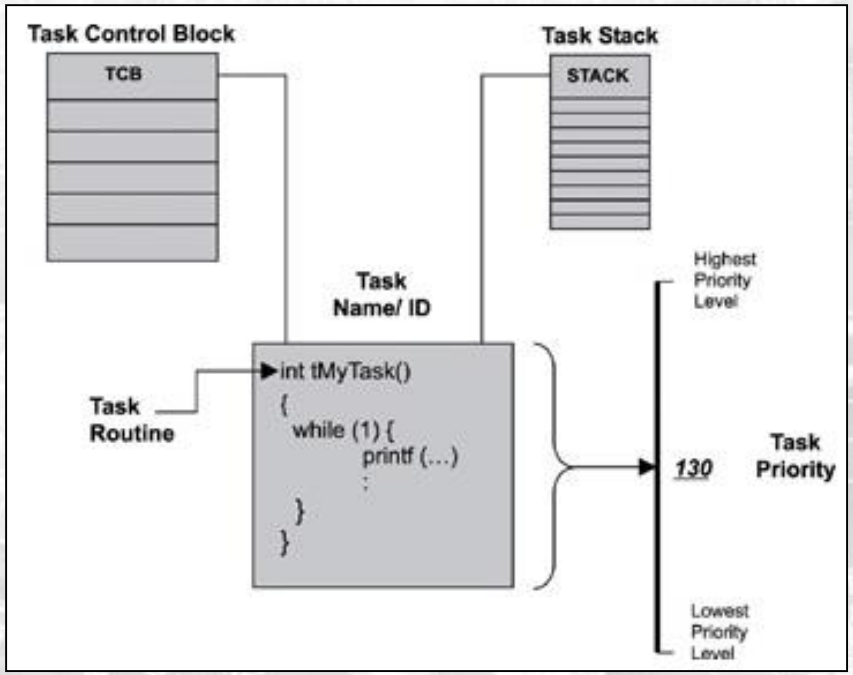
RTOS – Principales elementos

- El planificador del RTOS es el componente esencial y define la planificación de tareas en base a lo visto previamente.

- Adicionalmente el sistema puede presentar una serie de objetos (tales como tareas, primitivas de sincronización y comunicación, etc.) y servicios (interfaces, APIs, estándares como POSIX).

RTOS – Principales elementos

- Tarea: **hilo independiente de ejecución.**
  - Compite con otras tareas por tiempo de CPU.
  - Son planificables.
  - TCB y otras estructuras.
  - Interface para gestionarlas
- **Los sistemas embebidos requieren ser diseñados de manera concurrente (descomponer el sist. en pequeñas unidades de programa secuenciales).**



RTOS – Principales elementos

- Estados de ejecución de las tareas:

- Algunos sistemas definen estados adicionales
  - suspendida
  - demorada
  - pendiente
  - etc.

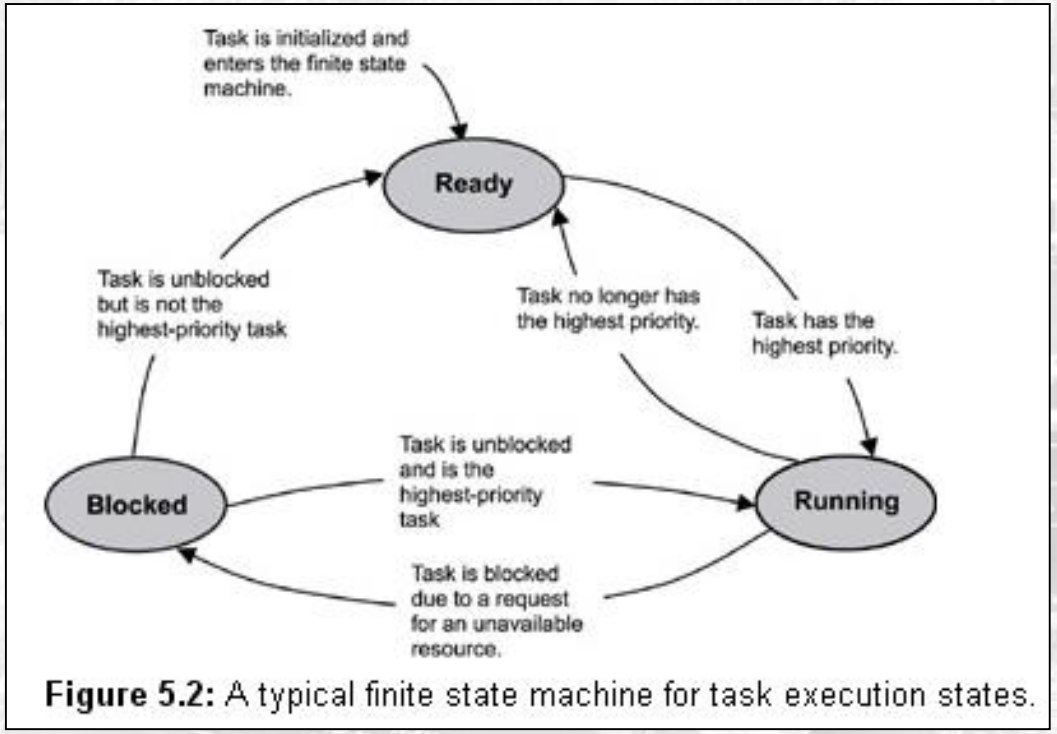


Figure 5.2: A typical finite state machine for task execution states.

RTOS – Principales elementos

- Además de las tareas definidas por el usuario, el RTOS define y crea ciertas tareas del sistema:
  - startup task
  - idle task
  - logging task
  - exception-handling task
  - debug agent task

- Adicionalmente se definen tareas para los servicios del sistema no provistos por el kernel (en presencia de RTOSes con microkernel).

Otros componentes

- Comunicación y sincronización entre tareas:
  - Semáforos (binarios, contadores, etc.)
  - Colas de mensajes, pipes
  - Registros de eventos
  - Variables de condición, monitores
- Eventos, excepciones e interrupciones
- Real time clocks (RTC), Relojes lógicos y de sistema
- Subsistema de I/O: BSPs y Device Drivers
- Gestión de memoria
- Soporte adicional:
  - Soporte de red / sistemas de archivo / shell de comandos

Otros componentes

- Real time clocks y System clocks:

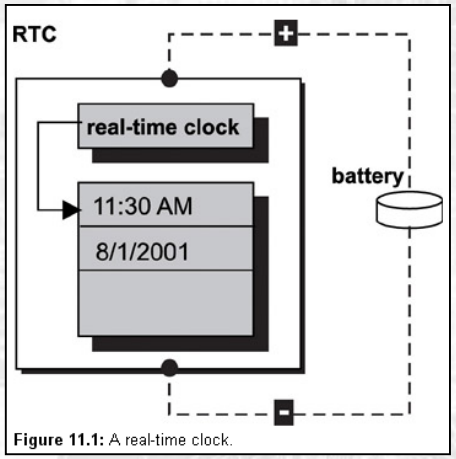
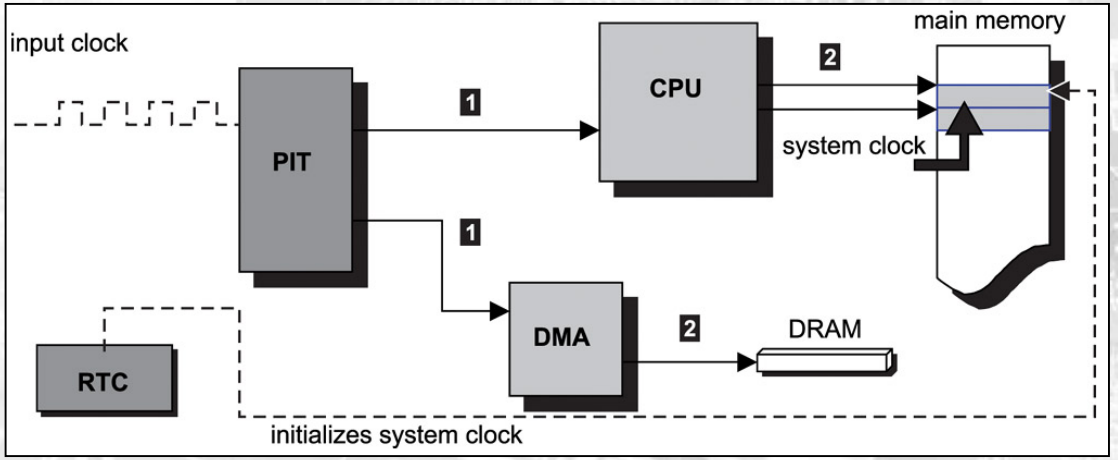
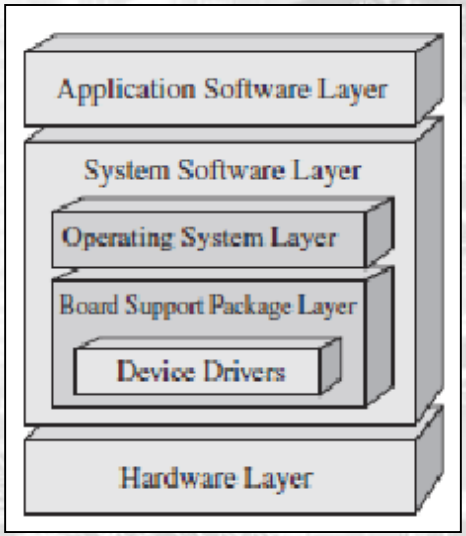


Figure 11.1: A real-time clock.



Otros componentes

- Board Support Packages (BSPs)
  - Comprende el software de soporte de la plataforma (placa, dispositivos, etc).
  - Separa los componentes independientes del hardware del target
  - Puede incluir:
    - Device drivers
    - Mapeo de pines y puertos
    - Mapa de memoria del procesador
    - Rutinas y librerías dependientes del hardware
    - Parámetros de incialización y configuración del target
    - Etc.



Otros componentes

- Sistemas de archivos de escritorio:
  - Ext2, Ext3 y Ext4 / Vfat
- Sistemas de archivo específicos p/memoria flash:
  - SquashFS (sólo lectura)
  - CRAMFS (sólo lectura)
  - JFFS2 (rw – recolección de basura, wear leveling, etc)
  - YAFFS2 (rw – recolección de basura, wear leveling, etc)
  - UBIFS (rw – rec. de basura, wear leveling, unsorted, etc)
- Ejecutan sobre una Flash Translation Layer (FTL) o un Memory Technology Device (MTD).
- Tradeoffs: performance (r/w), espacio utilizado, etc.

Requerimientos para RTOS

RTOS – Requerimientos

- La aplicación define los requerimientos del RTOS subyacente.
- Requerimientos más comunes:
  - Confiabilidad:
    - Para ejecutar durante largos períodos sin intervención humana.
    - Diversos grados de confiabilidad.
    - Se mide esta calidad no sólo sobre el RTOS, sino sobre todo el sistema en su conjunto.
  - Predicibilidad:
    - Respetar las restricciones temporales del sistema.
    - Tiempos de respuesta no sujetos a eventualidades (baja varianza).

RTOS – Requerimientos

- **Requerimientos más comunes:**
  - Performance:
    - **Necesaria para alcanzar los requerimientos temporales.**
    - **A mayor cantidad de deadlines, mayores requerimientos sobre el hardware (CPU) y la performance del RTOS.**
  - Compacidad:
    - **Ocupar poco espacio (menor ocupación estática y dinámica de memoria del RTOS en conjunto con las aplicaciones).**
    - **Menor requerimiento de memoria implica menores costos.**
  - Escalabilidad:
    - **Ajustarse a los requerimientos de la aplicación.**
    - **Facilidad para añadir y quitar componentes en función de las necesidades y de la configuración de hardware subyacente.**

¿Cuándo usar un RTOS?

- No hay respuestas absolutas.
- Lo más recomendable es mantener el diseño lo más simple posible.
  - Si el sistema realiza pocas tareas, puede resultar conveniente planificarlas de manera ad-hoc con un esquema Round-robin cíclico.
  - Aún en presencia de interrupciones, para sistemas simples es factible diseñar la aplicación directamente sobre el hardware.
  - Se debería utilizar un RTOS en la medida que se haga uso extensivo de los objetos y servicios que provee, o en sistemas que requieran la planificación de numerosas tareas (algunas críticas y otras no).

Eligiendo un RTOS

- Aspectos a tener en cuenta al momento de elegir un RTOS:
  - Soporte de procesadores.
  - Características de tiempo real (planificación, etc).
  - Costos y tipo de licencia.
  - Uso de memoria.
  - Drivers y soporte de componentes.
  - Soporte técnico.
  - Compatibilidad con las herramientas de desarrollo.
  - Preferir versiones de código abierto (modificar aspectos problemáticos, mayor facilidad para portarlo a otras plataformas).

Referencias

- Barr, M., Massa, A. Programming Embedded Systems: With C and GNU Development Tools, 2nd Edition. O'Reilly Media. 2006. ISBN: 978-0596009830. Capítulo 10.
- Li, Q., Yao, C. Real-time concepts for Embedded Systems. CMP. 2003. ISBN: 978-1578201242. Capítulos 4 al 14 y 16.
- Noergaard, T. Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers. Newnes. 2005. ISBN: 978-0750677929. Capítulo 9.
- Simon, D. An Embedded Software Primer. Addison-Wesley Professional. 1999. ISBN: 978-0201615692. Capítulos 6 y 7.