Sistemas Operativos en Tiempo Real (Real Time Operative Systems) en sistemas embebidos

Carlos Becker

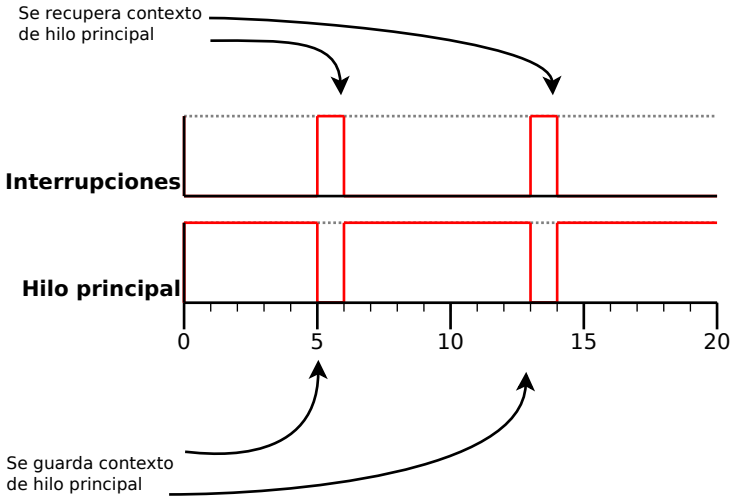
carlosbecker@gmail.com

Junio 2009

Sistemas de hilo único

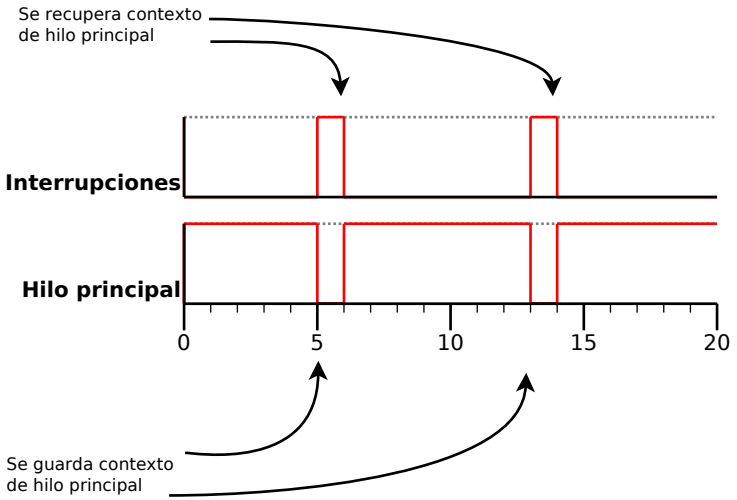
Sistemas de hilo único

- Único camino de ejecución, excepto por interrupciones



Sistemas de hilo único

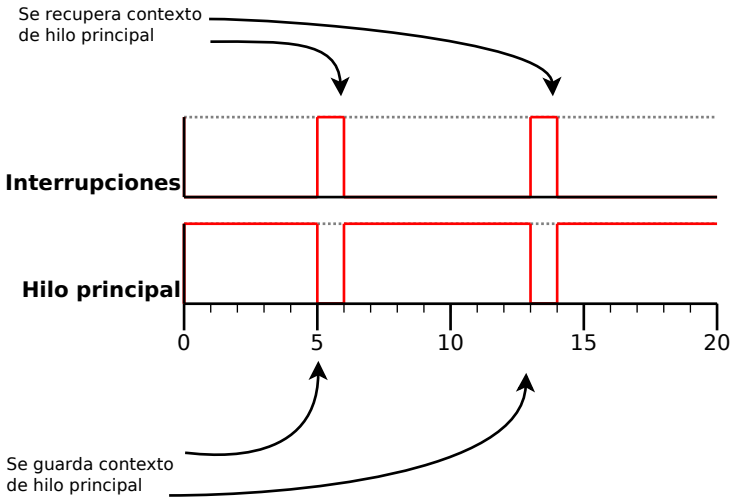
- Único camino de ejecución, excepto por interrupciones



- Stack único (excepto por interrupciones/excepciones)

Sistemas de hilo único

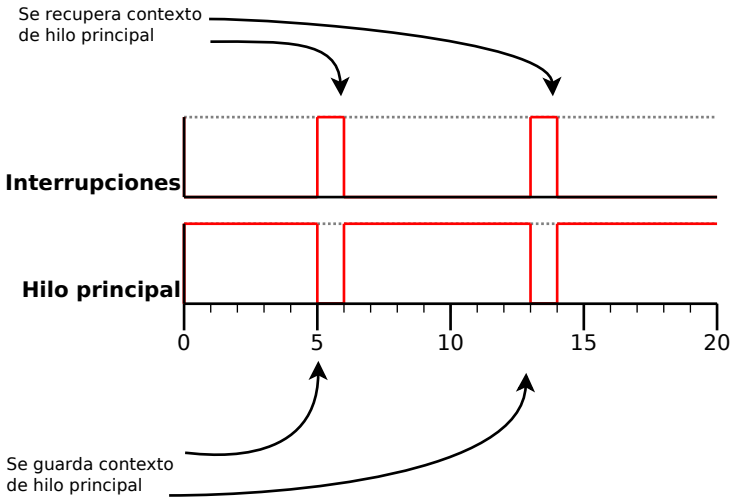
- Único camino de ejecución, excepto por interrupciones



- Stack único (excepto por interrupciones/excepciones)
- Sistemas complejos difíciles de mantener y/o extender

Sistemas de hilo único

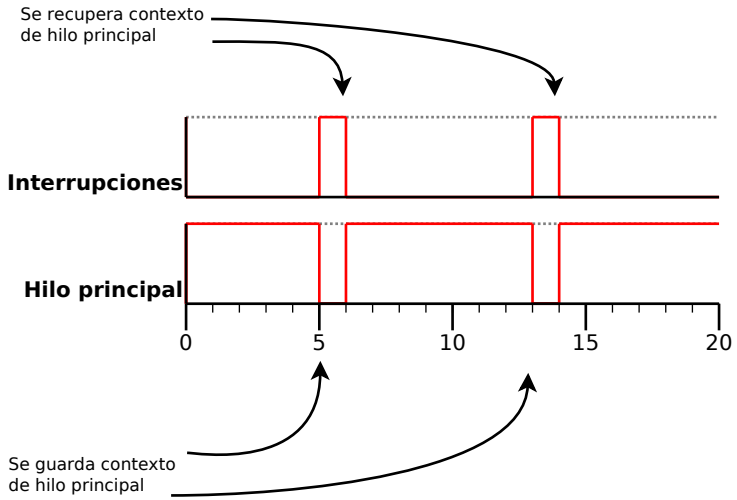
- Único camino de ejecución, excepto por interrupciones



- Stack único (excepto por interrupciones/excepciones)
- Sistemas complejos difíciles de mantener y/o extender
- Resulta difícil controlar la precisión en las temporizaciones

Sistemas de hilo único

- Único camino de ejecución, excepto por interrupciones



- Stack único (excepto por interrupciones/excepciones)
- Sistemas complejos difíciles de mantener y/o extender
- Resulta difícil controlar la precisión en las temporizaciones
- Existen algunas alternativas para simular concurrencia como ProtoThreads. Limitaciones

Ejemplo de hilo único


```
void main(void)
{
    for (;;)
    {
        if ( datosSeriePendientes() ) {
            procesarDatosSerie();
        }

        if ( pasaronMasDe500mseg() ) {
            titilarLedActividad();
        }
    }
}
```

- Funciones deben cooperar entre si
- La programación se vuelve compleja para sistemas grandes

Sistema Multi Hilado en un mismo procesador

Tarea

- Una tarea es un camino de ejecución a través del espacio de direcciones.
- Cada tarea cumple una función específica

Sistema Multi Hilado en un mismo procesador

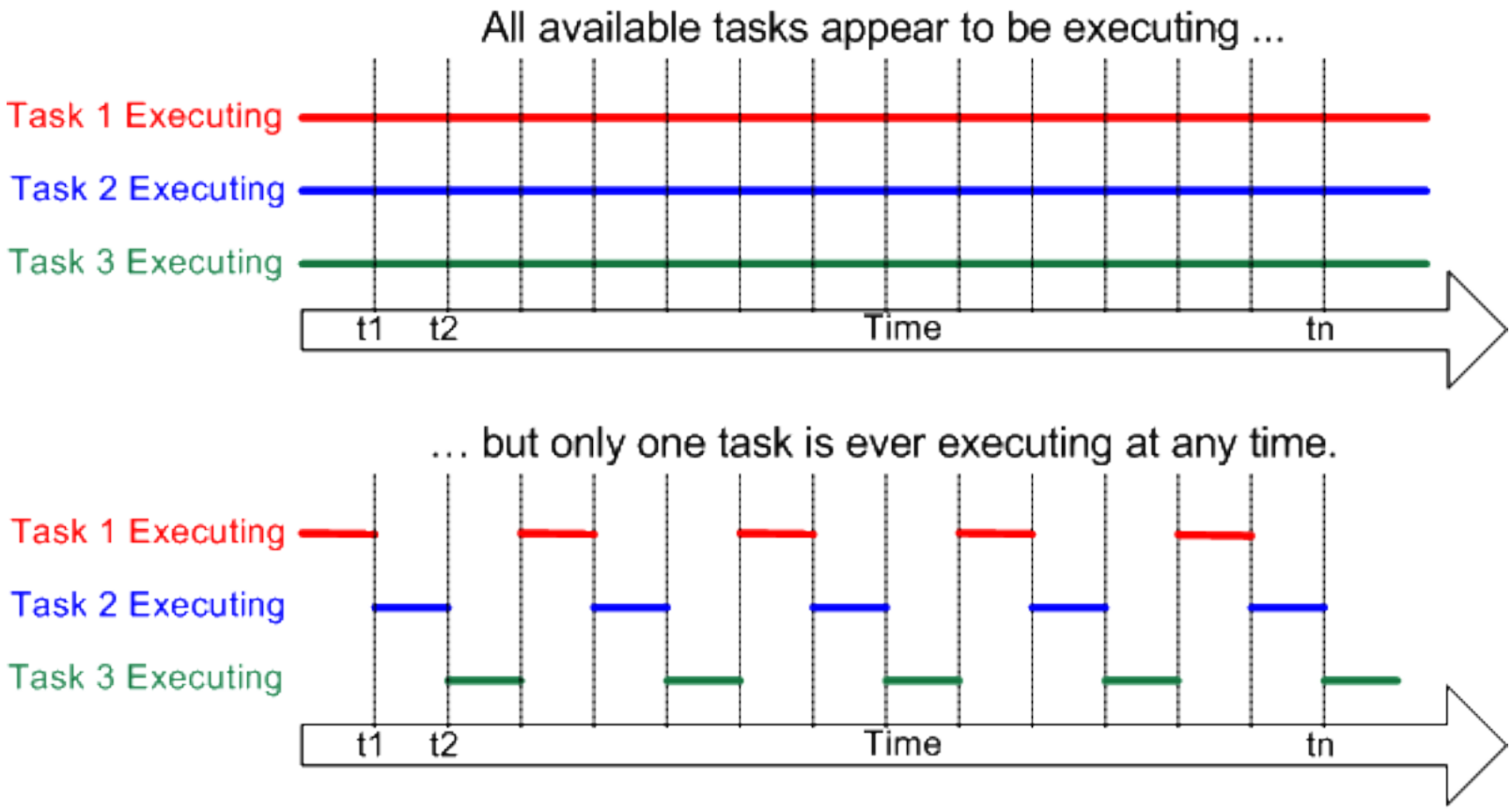


Figura: Sistema multitarea - http://FreeRTOS.org

Sistema Multi Hilado en un mismo procesador

http://zazzle.com

Sistemas Multi Hilados en un mismo procesador

Sistemas Multi Hilados en un mismo procesador

- Simulan un CPU propio para cada tarea/hilo

Sistemas Multi Hilados en un mismo procesador

- Simulan un CPU propio para cada tarea/hilo
- Cada hilo tiene una prioridad asignada

Sistemas Multi Hilados en un mismo procesador

- Simulan un CPU propio para cada tarea/hilo
- Cada hilo tiene una prioridad asignada
- Mantenimiento y extensibilidad más sencillos en sistemas grandes

Sistemas Multi Hilados en un mismo procesador

- Simulan un CPU propio para cada tarea/hilo
- Cada hilo tiene una prioridad asignada
- Mantenimiento y extensibilidad más sencillos en sistemas grandes
- Se pueden temporizar eventos de manera más precisa y sencilla, aún en sistemas complejos

Sistemas Multi Hilados en un mismo procesador

- Simulan un CPU propio para cada tarea/hilo
- Cada hilo tiene una prioridad asignada
- Mantenimiento y extensibilidad más sencillos en sistemas grandes
- Se pueden temporizar eventos de manera más precisa y sencilla, aún en sistemas complejos
- En general requieren más memoria (RAM) que un sistema de hilo único

Sistemas Multi Hilados en un mismo procesador

- Simulan un CPU propio para cada tarea/hilo
- Cada hilo tiene una prioridad asignada
- Mantenimiento y extensibilidad más sencillos en sistemas grandes
- Se pueden temporizar eventos de manera más precisa y sencilla, aún en sistemas complejos
- En general requieren más memoria (RAM) que un sistema de hilo único
- Los cambios de contexto no son instantáneos.

Tareas

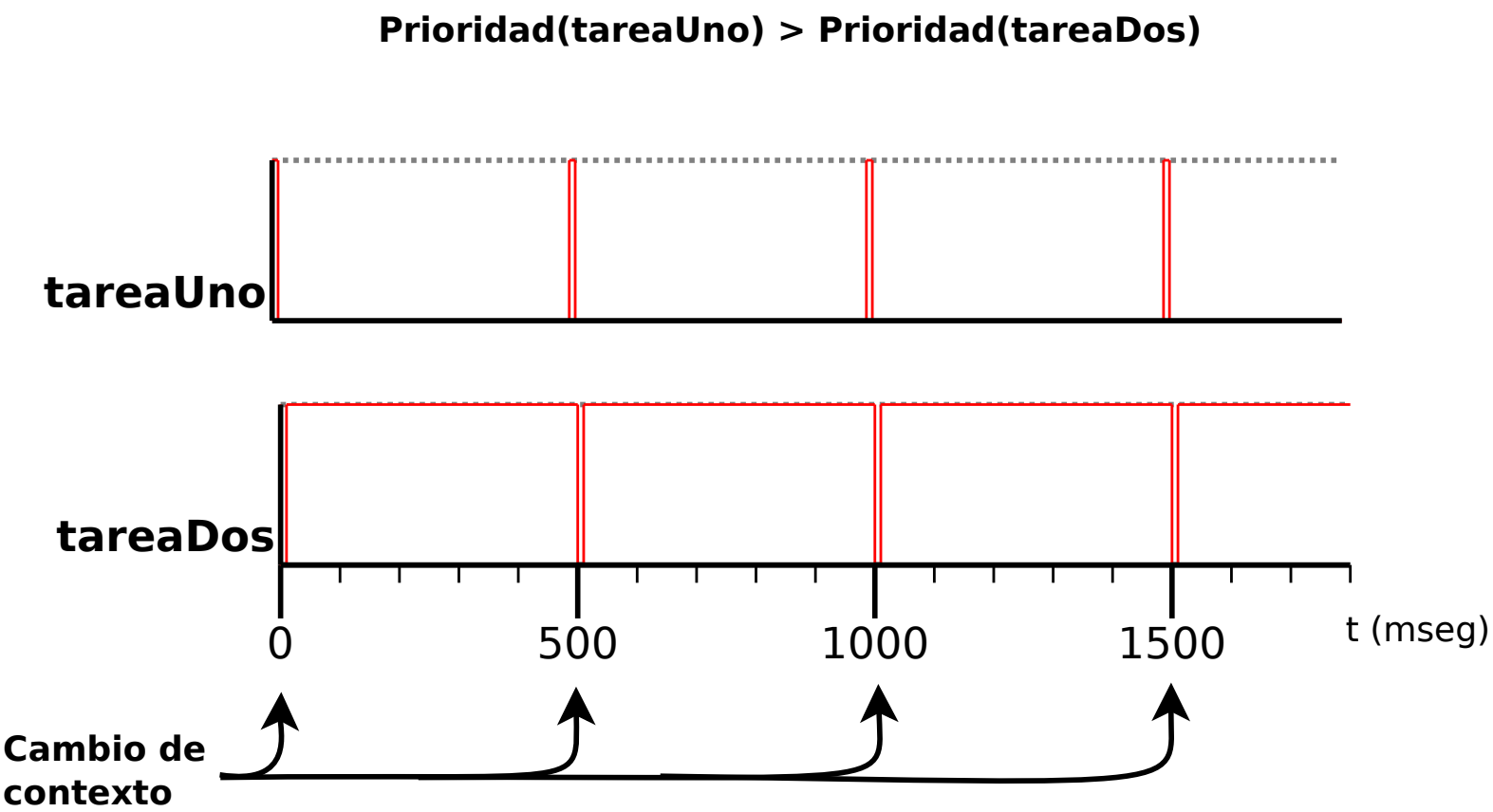
Sistema operativo guarda

- Contador de programa, puntero a stack y todos los registros del CPU
- Estado actual de la tarea
  - Ejecutándose
  - Bloqueada (espera E/S o recurso)
  - Detenida
  - ...
- Prioridad
- Recursos del SO asociados a dicha tarea

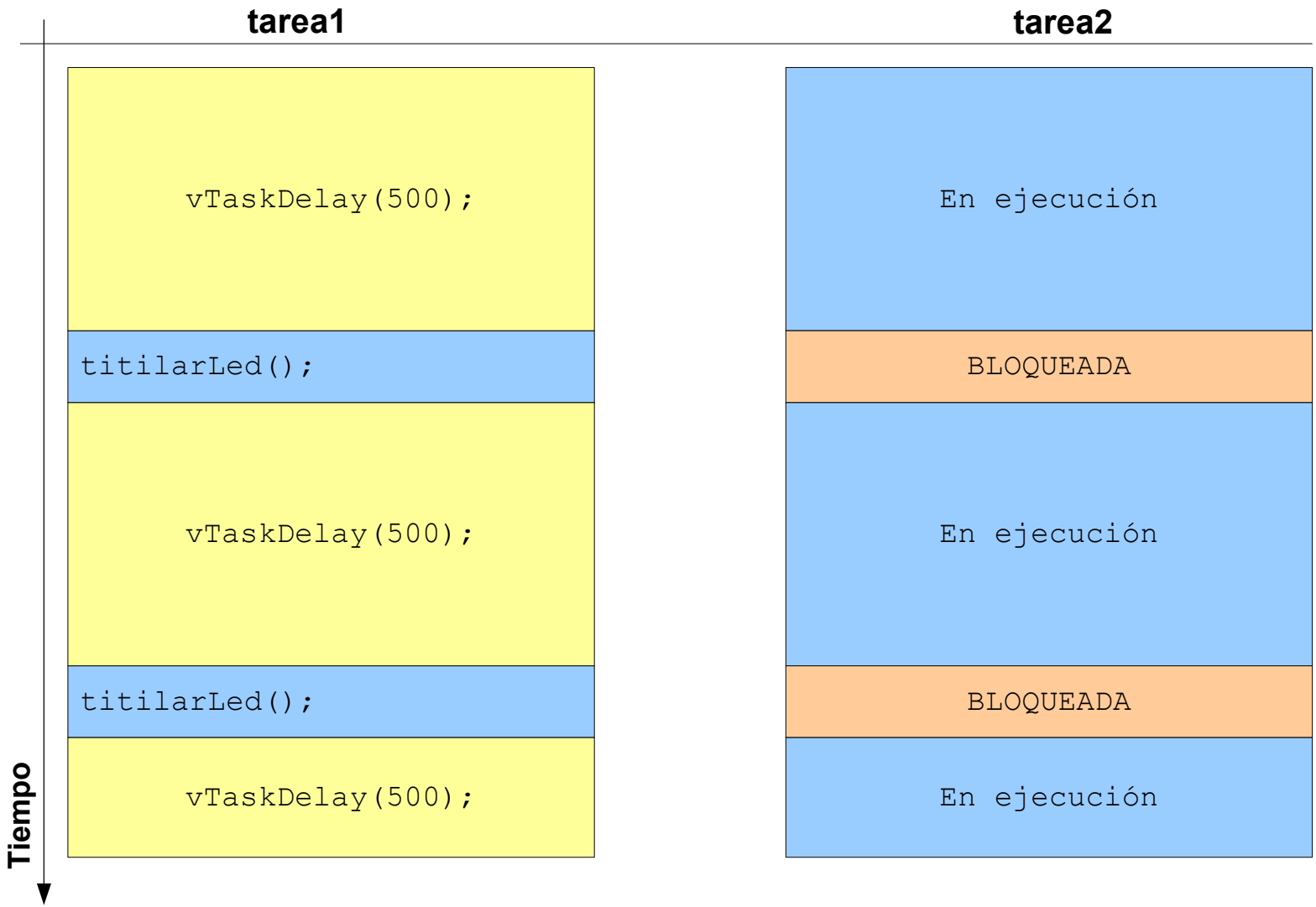
Tareas independientes


```
#include <RTOS.h>

void tareaUno(void *p)
{
    for (;;) {
        titilarLed();
        vTaskDelay(500);
    }
}

void tareaDos(void *p)
{
    for (;;) {
        recibirDatosSerie();
        procesarDatosSerie();
    }
}

void main(void)
{
    registrarTarea( &tareaUno, "uno", PRIORIDAD2 );
    registrarTarea( &tareaDos, "dos", PRIORIDAD1 );

    iniciarRTOS();
}
```

Tareas independientes - Diagrama de tiempos



Tareas independientes - Diagrama de tiempos



Sistemas apropiativos y cooperativos

Sistema cooperativo

- El SO no puede detener una tarea arbitrariamente
- Tareas deben ceder control al SO explícitamente
- Esto último implica, por lo general, valores de latencia altos al responder a eventos de tiempo real

Sistema apropiativo

- SO puede detener una tarea arbitrariamente, reanudando la ejecución de otra según lo considere necesario
- Tareas no necesitan ceder control al SO
- Se reduce de manera importante el tiempo de respuesta a eventos de tiempo real: RTOS

Comunicación entre tareas

Recursos provistos por el SO. Permiten que una tarea pueda bloquear esperando por un evento.

Recursos comunes para comunicación

- Memoria compartida
- Semáforos / Mutexes
- Colas de mensajes

Qué es FreeRTOS

- RTOS de código abierto
- Múltiples ports a diferentes arquitecturas/micros
- Totalmente gratuito
- http://www.FreeRTOS.org

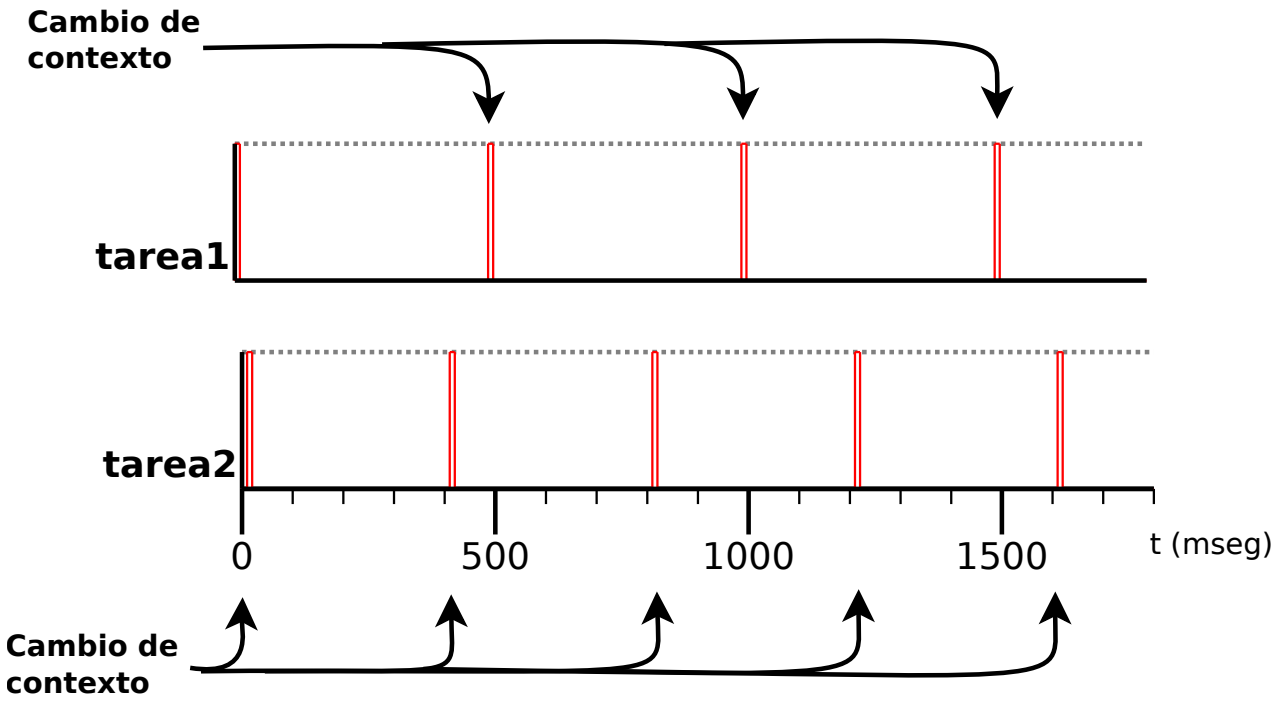
Recursos provistos

- Tareas
- Co-rutinas
- Planificador Round-Robin
- Colas de mensajes
- Semáforos / Mutexes

Ejemplo FreeRTOS

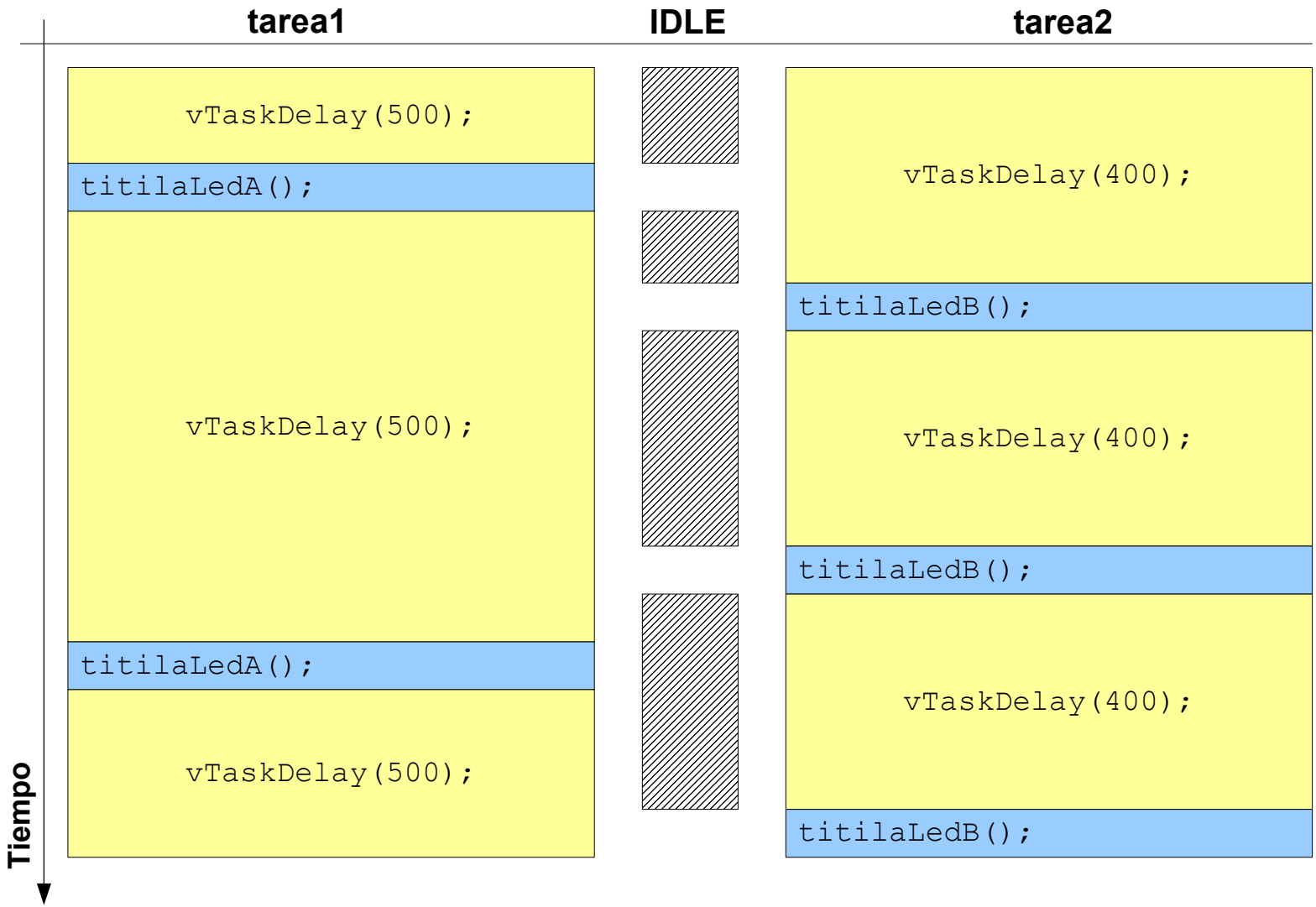

```
#include <FreeRTOS.h>
#include <task.h>

void tarea1( void *p )
{
    for (;;)
    {
        titilarLedA();
        vTaskDelay(500);
    }
}

void tarea2( void *p )
{
    for (;;)
    {
        titilarLedB();
        vTaskDelay(400);
    }
}

void main()
{
    xTaskCreate( &tarea1, "tarea1", /* stack */ 200, NULL, 2, NULL );
    xTaskCreate( &tarea2, "tarea2", /* stack */ 200, NULL, 1, NULL );

    vTaskStartScheduler();
}
```

Ejemplo FreeRTOS - Diagrama de tiempos



- $Prioridad(tarea1) > Prioridad(tarea2)$
- ¿Qué sucede cuando ninguna tarea se está ejecutando?

Ejemplo FreeRTOS - Diagrama de tiempos



Problemas de concurrencia

Problemas de concurrencia

- En general los recursos de hardware no están pensados para ser accedidos de manera concurrente

Problemas de concurrencia

- En general los recursos de hardware no están pensados para ser accedidos de manera concurrente
- Se debe implementar la protección necesaria para evitar problemas de acceso

Ejemplo de conflicto debido a concurrencia


```
void prenderLed( int led )
{
    PORTB = PORTB | (1<<led);
}

void tareaA( void *p )
{
    for (;;) {
        vTaskDelay(200);
        prenderLed( 1 );
    }
}

void tareaB( void *p )
{
    for (;;) {
        vTaskDelay(100);
        prenderLed( 2 );
    }
}
```

El problema surge de la no atomicidad de ciertas sentencias u operaciones.

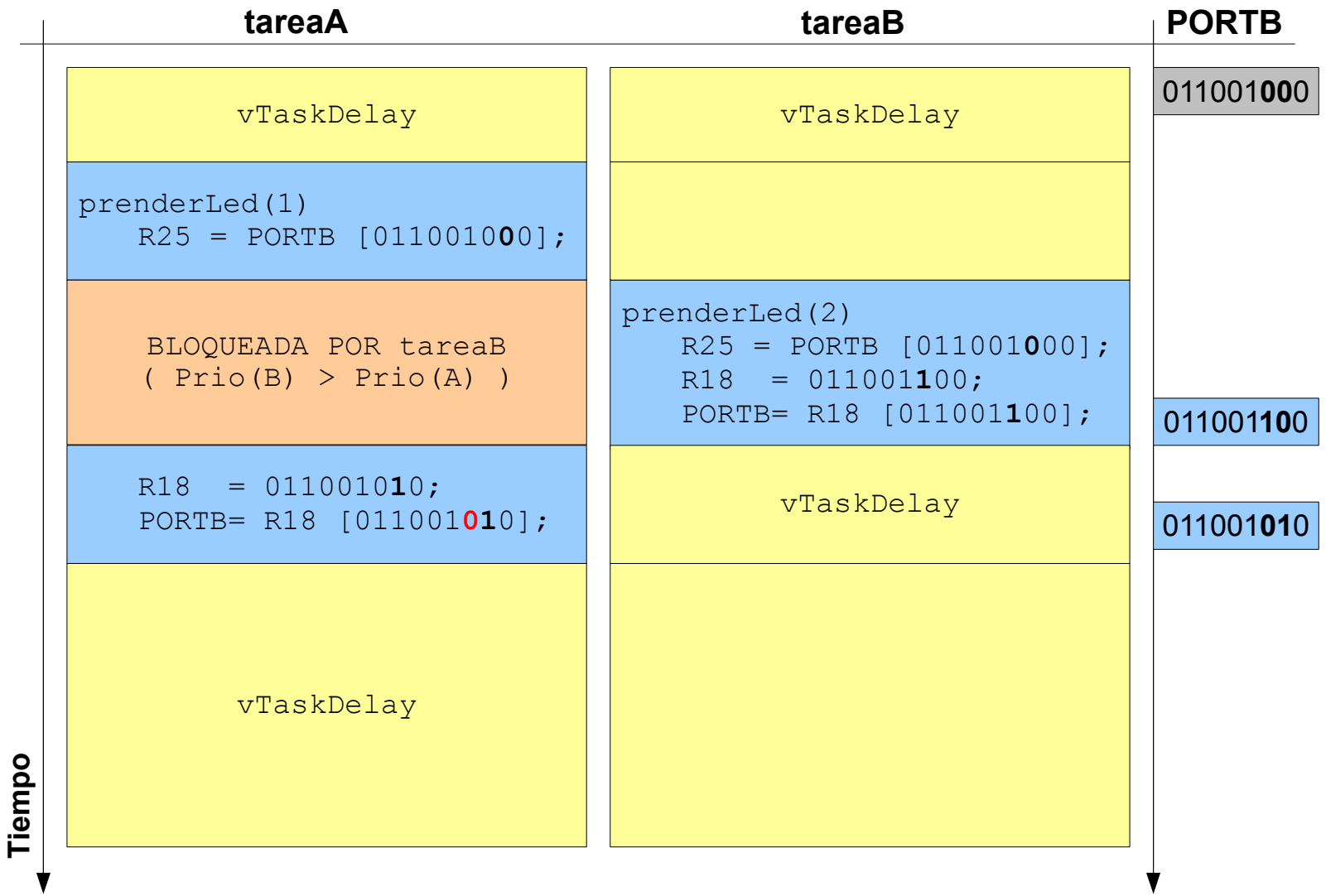
Ejemplo de conflicto debido a concurrencia

Ejemplo para AVR 8-BIT

```
void prenderLed( int led )
{
     d6:     in    r25, 0x05          ; r25 = PORTB
     d8:     ldi   r18, 0x01          ; r18 = 0x01
     da:     ldi   r19, 0x00          ; r19 = 0x00
     dc:     rjmp  .+4                ; salta a 0xe2
     de:     add   r18, r18           ; r18 <<= 1
     e0:     adc   r19, r19           ; r19 <<= 1
     e2:     dec   r24                ; r24--
     e4:     brpl  .-8                ; a 0xde si r24 > 0
     e6:     or    r25, r18           ; r25 |= r18
     e8:     out   0x05, r25          ; PORTB = r25
     ea:     ret
}
```

El problema surge de la no atomicidad de ciertas sentencias u operaciones.

Ejemplo de conflicto debido a concurrencia

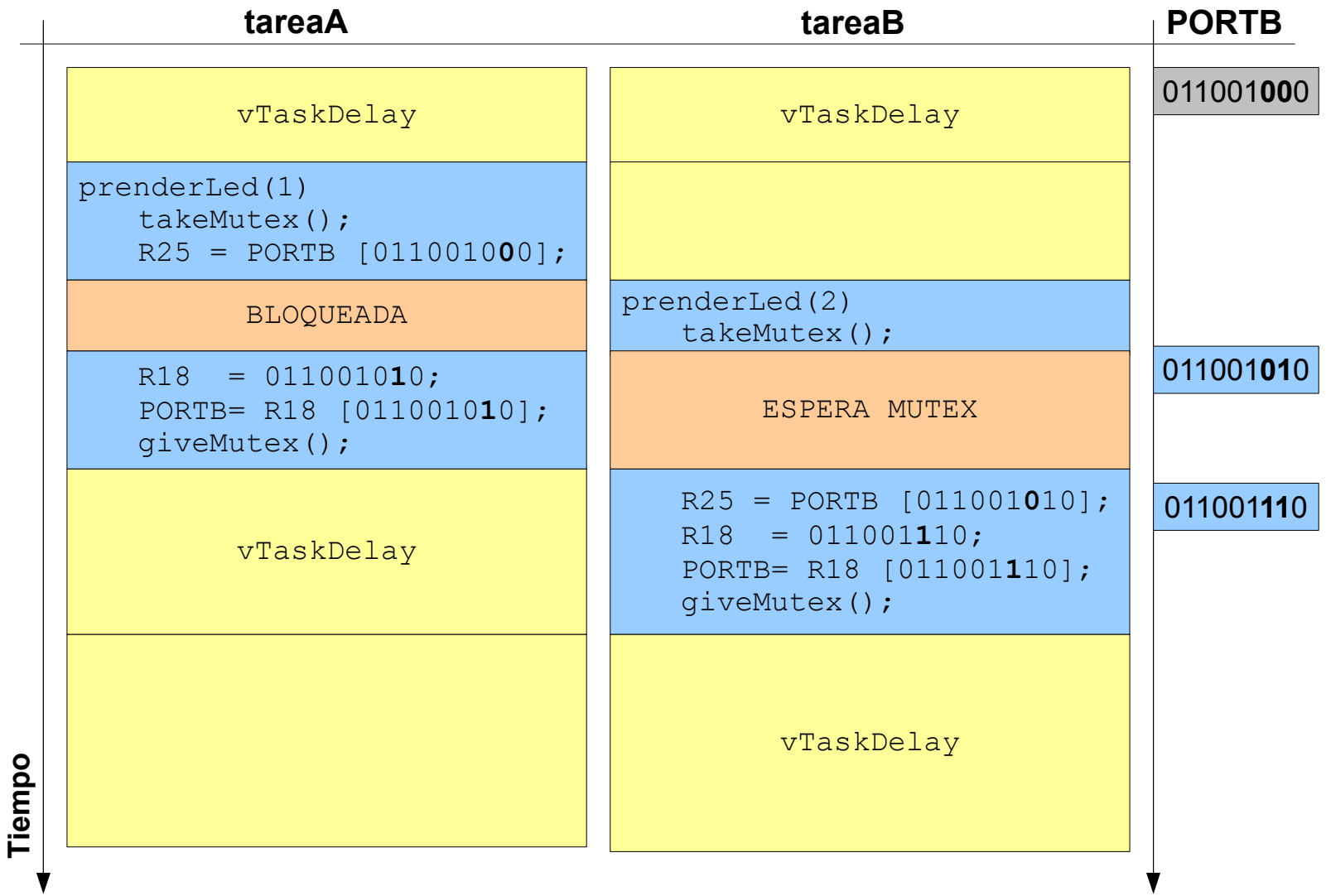


Solución: Mutexes


```
void prenderLed( int led )
{
    takeMutex();
        PORTB = PORTB | (1<<led);
    giveMutex();
}

void mandarByte( unsigned char b )
{
    takeMutex();
        REGTX = b;
        while ( REGCTRL & (1<<SENDOK) )
            ;
    giveMutex();
}
```

La ejecución se bloquea en el caso que otra tarea ya haya tomado el mutex. Los mutex son recursos que provee el SO.

Ejemplo de conflicto debido a concurrencia



Solución: Ejemplo con FreeRTOS


```
  #include <FreeRTOS.h>
  #include <task.h>
  #include <semphr.h>

  xSemaphoreHandle    mutex1, mutex2;

void inicializar(void)
{
    mutex1 = xSemaphoreCreateMutex();
    mutex2 = xSemaphoreCreateMutex();
}

void prenderLed( int led )
{
    xSemaphoreTake( mutex1, portMAX_DELAY );
        PORTB = PORTB | (1<<led);
    xSemaphoreGive( mutex1 );
}

void mandarByte( unsigned char b )
{
    xSemaphoreTake( mutex2, portMAX_DELAY );
        REGTX = b;
        while ( REGCTRL & (1<<SENDOK) )
            ;
    xSemaphoreGive( mutex2 );
}
```

Misma función- múltiples tareas

Misma función- múltiples tareas

- Para identificar a cada tarea se utiliza el puntero que lleva como argumento la función correspondiente

Misma función- múltiples tareas

- Para identificar a cada tarea se utiliza el puntero que lleva como argumento la función correspondiente
- Se crean tantas tareas como se necesiten, pudiendo compartir la misma prioridad o no

Misma función- múltiples tareas

- Para identificar a cada tarea se utiliza el puntero que lleva como argumento la función correspondiente
- Se crean tantas tareas como se necesiten, pudiendo compartir la misma prioridad o no
- Con modificar dicha función se modifica el comportamiento de todos los hilos:
  - Se diseña y piensa una sola tarea, extrapolando a N tareas mediante el RTOS
  - Es muy importante tener en cuenta posibles problemas de concurrencia

Misma función- múltiples tareas


```
typedef struct {
    int    num;
    /* otros datos que puedan ser necesarios */
} infoTarea;

/* Se pasa puntero a estructura de información */
void funcTarea( infoTarea *p )
{
    for (;;) {
        vTaskDelay(100);
        titilarLed( p->num ); //depende de la tarea
    }
}

infoTarea info1, info2, info3;

void main()
{
  info1.num = 1; //llenamos datos
  info2.num = 3;
  info3.num = 4;

  xTaskCreate( &funcTarea, "tarea1", /* stack */ 200, &info1, 1, NULL );
  xTaskCreate( &funcTarea, "tarea2", /* stack */ 200, &info2, 1, NULL );
  xTaskCreate( &funcTarea, "tarea3", /* stack */ 200, &info3, 1, NULL );

  vTaskStartScheduler();
}
```

Tarea de monitoreo - Watchdog

Tarea de monitoreo - Watchdog

- Tarea de menor prioridad que indique el correcto funcionamiento del sistema

Tarea de monitoreo - Watchdog

- Tarea de menor prioridad que indique el correcto funcionamiento del sistema
- Es posible utilizar esta tarea para refrescar el WatchDog (si se posee uno)

Tarea de monitoreo - Watchdog

- Tarea de menor prioridad que indique el correcto funcionamiento del sistema
- Es posible utilizar esta tarea para refrescar el WatchDog (si se posee uno)
  - Las demás tareas no deben impedir la ejecución de la tarea de monitoreo por un tiempo prolongado
  - Es aconsejable evaluar variables que permitan determinar si el sistema está funcionando correctamente antes de refrescar el WatchDog.

```
/**
 * Tarea que posee la prioridad más baja
 */
void tareaMonitor( void *p )
{
    for (;;) {
        vTaskDelay(100);
        clearWatchDog(); // limpiamos watchdog
        titilarLed(); // indicamos que el aparato funciona
    }
}
```

Ejemplo Real

Correo de Voz Nexo NVM-4E

Ejemplo Real

Correo de Voz Nexo NVM-4E

- Procesamiento de audio en tiempo real y grabación/reproducción

Ejemplo Real

Correo de Voz Nexo NVM-4E

- Procesamiento de audio en tiempo real y grabación/reproducción
- Conexión Ethernet 10/100Mbps

Ejemplo Real

Correo de Voz Nexo NVM-4E

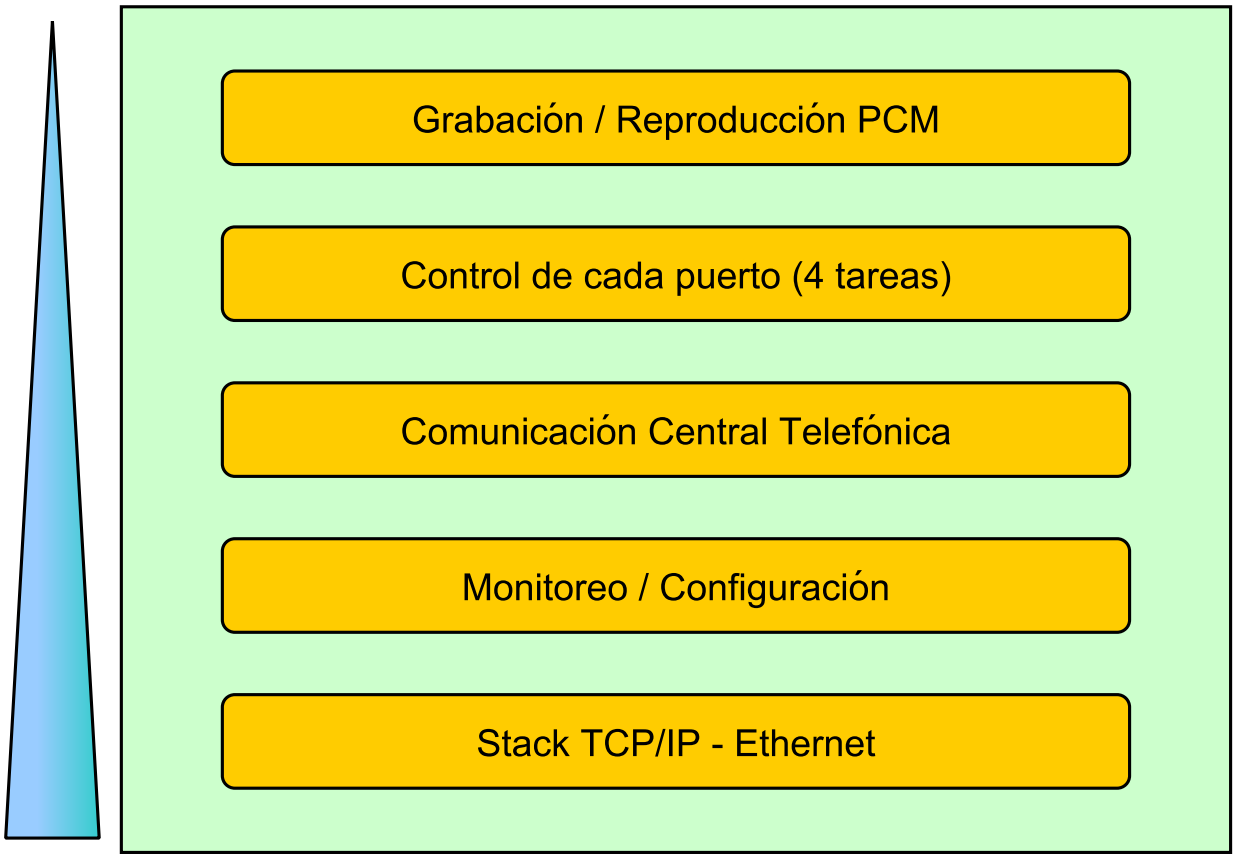
- Procesamiento de audio en tiempo real y grabación/reproducción
- Conexión Ethernet 10/100Mbps
- Conexión USB

Ejemplo Real

Correo de Voz Nexo NVM-4E

- Procesamiento de audio en tiempo real y grabación/reproducción
- Conexión Ethernet 10/100Mbps
- Conexión USB
- Cuatro puertos telefónicos activos en simultáneo

16 Tareas/Hilos



Resumen de tareas, prioridades y uso de stack.

```
--------- STACK WATERMARK =========-
Tasks: 16

Name     Prio   UsedStack/AvailStack
-----------------------------------
lwIP     3      112/900
main     2      103/200
webSvr   3      127/190
naudio   3      73/240
EDBG     2      40/130
TKC      4      50/100
tek      4      55/120
pcm      6      63/170
pleds    2      42/100
port0    5      113/210
port1    5      114/210
port2    5      103/210
port3    5      102/210
mon      4      97/180
ETH_INT  4      48/150
eth      1      36/75
-================================-
```

Otros Sistemas Operativos

- QNX (RTOS)
- Linux (No en tiempo real)
- RTLinux (RTOS)
- VxWorks
- Windows CE
- μC/OS-II
- DSPBios

Conclusiones

- Un RTOS permite organizar tareas de manera lógica y sencilla, asignando una prioridad a cada uno de ellos.
- *Tiempo real* significa que el Sistema Operativo garantiza una latencia reducida para los eventos e interrupciones, dado un diseño adecuado del software y las tareas.

Conclusiones - Ventajas y desventajas

Ventajas

- Reducción del tiempo de desarrollo
- Refuerza conceptos de modularización
  - Provee de métodos seguros de comunicación entre tareas
  - Asegura la independencia entre tareas
- Facilita la extensibilidad y verificación del diseño

Desventajas

- Desarrollar software para un RTOS requiere de experiencia y planeamiento cuidadoso:
  - Controlar acceso a recursos utilizados por más de una tarea
- Requieren más memoria RAM que un sistema sin RTOS