Formulacion de una Métrica en la evaluación de los Sistemas Operativos en Tiempo Real para sistemas embebidos.

Ing. Marcelo Romeo[1], Ing. Eduardo Martínez[2], Ing. Rodolfo Recanzone[3], Prof. Frédéric Amiel[4]
[1] Universidad de Belgrano, UNSAM, UTN, [2] Universidad de Belgrano, [3]FCEIA – Universidad Nacional de Rosario , [4] ISEP – París

Abstract—The fundamental requirements to be successful for a project of a digital system are:

• Appropriate technological solution to the problem

• The quick time-to-market solution.

On the first point, there are many tools and methods to find a suitable solution cost / performance

On the second point, the market offers low cost assembled and tested boards which are efficient for small productions or good basis for developing a dedicated circuit for large productions.

Today every circuit must be accompanied by a program that will command it. This is where a bottleneck for getting a product in the shortest possible time may occur.

For this reason is that they have developed real-time operating systems that serve as solid basis for the specific program that will solve the problem.

This is why Real-Time Operating Systems were developed, serving as a solid basis for the specific program that will solve the problem.

The aim of this paper is to propose some methods to evaluate different RTOS, and provide the designer with tools that help him to select the most appropriate for his project.

Keywords: Real Time, RTOS, embedded, benchmarks.

I. INTRODUCCION

Existen numerosas técnicas para escribir buen software sin el uso de un sistema operativo y si el sistema es sencillo, esas técnicas proveerán una solución apropiada.

En sistemas más complejos el uso de un sistema operativo, disminuirá el tiempo de desarrollo y mejorará la calidad del producto final. El punto de cruce es particularmente subjetivo.

¿Porqué emplear un kernel.? [1]

- **Minimización de *time-to-market*:** De la misma manera que para disminuir los tiempos de desarrollo del hardware de un proyecto se emplean placas pre-armadas, podemos también emplear un *kernel* que nos disminuya el tiempo de desarrollo del software.
- **Inicialización:** En primer lugar, tendremos buena parte (en algunos casos, toda) de la tediosa y compleja inicialización del sistema resuelta.
- **División del proyecto en tareas:** Además podremos dividir el proyecto en tareas (*tasks)* sencillas de escribir y de probar y controlar las interdependencias entre las mismas. Esas tareas serán modulares ya que tendrán propósitos bien definidos. Las tareas podrán estar constituidas por varios hilos ***(threads)*** que son procesos livianos,
- **Trabajo en equipo:** Las tareas tendrán claras variables de entrada y salida, por lo que se facilitará la división del trabajo en un equipo de desarrolladores.
- **Eficiencia:** El uso de un *kernel* permite que el software sea completamente *event-driven*, de forma de evitar encuestas innecesarias sobre los requerimientos de atención de dispositivos o tareas sin actividad.
- **Reutilización del código:** La modularidad permite reutilizar código en varios proyectos.
- **Mantenimiento:** La modularidad permite aislar fallas y agregar funciones en forma sencilla.
- **Seguridad:** El empleo de un ***kernel*** con adecuadas normas de seguridad incrementará la seguridad del sistema haciendo menos probable un crash del conjunto.

II. OBJETIVO

Existen desarrolladores experimentados que sostienen que un RTOS no es imprescindible cuando se tiene un proyecto bien estructurado y con reglas de buen diseño. Contrariamente otros diseñadores sostienen que un RTOS puede ser utilizado aún en tareas sencillas para hacer experiencia y armar bibliotecas, ambas necesarias para proyectos de mayor envergadura.

Suponiendo que hemos decidido que nuestro proyecto requiere el uso de un sistema operativo en tiempo real ***(RTOS),*** deberemos definir cuál utilizar.

Siempre aparece la tentación de desarrollar nuestro propio RTOS, lo cual es desaconsejado salvo que se trate de una aplicación muy específica no cubierta por los RTOS estándares.

Es una decisión sin reglas generales, propias de cada tipo de proyecto y algo subjetivas. No será lo mismo la selección de un *kernel* para una puerta automática de ascensor que para un router inalámbrico.

Deberemos también comparar los beneficios que nos provee el uso de un RTOS frente a la sobrecarga ***(overhead)*** que introduce en el sistema.

En este trabajo, deseamos presentar algunos lineamientos que nos ayuden en el análisis, proveyendo algunos puntos a tener en cuenta en la selección del sistema operativo que mejor se perfile para nuestra aplicación. [2]

III. PLANTEO

Reafirmamos que el uso de un *RTOS* disminuirá el tiempo de desarrollo del software pues deberá proveernos de:

- Administración de las tareas, conmutando la tarea activa según reglas a definir en cada caso.

- Comunicaciones, resolviendo el manejo de dispositivos de comunicación serie, USB y frecuentemente Ethernet.
- Asignación de recursos a cada tarea (memoria de programa y datos, periféricos a emplear, etc.)
- Servicios
- Confiabilidad

Aislando a las tareas de esos detalles por medio de la sencilla invocación de un programa de interfaz con la aplicación ***(API).***

IV. PUNTOS A TENER EN CONSIDERACIÓN EN LA SELECCIÓN DE UN RTOS

a) SEGURIDAD Y CONFIABILIDAD:

La confiabilidad de un RTOS comienza en la robustez de la API. En los más estrictos, se verifica la consistencia de los parámetros pasados a la misma para que no haya errores en la invocación y que el programador desee emplear una tarea inexistente o pretenda emplear la CPU un tiempo mayor al disponible. Esas buenas prácticas preventivas comenzaron con al producirse catástrofes debidas a errores de software que en un comienzo eran inimaginables.

Existen regulaciones internacionales [3] [4] [5] que marcan estrictas pautas que debe cumplir el software en el diseño, desarrollo y prueba (*testing*) de acuerdo con el compromiso de vida de aquellos que se puedan ver afectados por el software (un avión, un equipo electromédico, un lavarropas, etc.).

Existen RTOS que ya han sido certificados y facilitan la aprobación de todo el software por los organismos reguladores.

Dichos RTOS son más costosos, por lo que su uso será recomendable solamente en los casos en que sean imprescindibles.

b) FACILIDAD DE USO

Debe estar adecuadamente documentado, con ejemplos de uso que permitan utilizar sencillamente la API, que es la puerta de entrada de nuestras tareas al mundo del RTOS.

c) POTENCIA CONSUMIDA.

En todos los sistemas operativos se implementa lo que en Windows es el "Proceso Inactivo del Sistema" y que puede emplearse para pasar a un modo de bajo consumo de forma de disminuir la potencia total consumida.

d) COMPORTAMIENTO EN TIEMPO REAL

En una encuesta realizada a desarrolladores de sistemas embebidos, el comportamiento en tiempo real fue el ítem que mayor exigencia requirió del 59% de los interrogados [2].

Si bien parece ser un parámetro trascendente no existe uniformidad de definiciones al respecto ni mucho menos como medirlo.

¿Qué es tiempo real?: Un sistema que opera en tiempo real es aquél que interactúa con un entorno con dinámica determinística, es decir que dadas las mismas entradas y estado inicial, recorre los mismos estados intermedios para llegar siempre al mismo estado final, generando las mismas salidas y preferiblemente en el mismo tiempo, de acuerdo con los conceptos de predictibilidad, estabilidad, controlabilidad y alcanzabilidad.

Tomemos como ejemplo el airbag de un automóvil. El tiempo de actuación no deberá depender de factores de menor prioridad como por ejemplo, estar sintonizando la radio del mismo. Se estipula como condición de diseño que deberá responder como máximo en 200 milisegundos y se deberá diseñar el sistema para que en cualquier condición responda en ese tiempo máximo incluyendo la sobrecarga que introduzca el RTOS. Los 200 ms máximos será nuestro entorno de Tiempo Real.

Parecería que todo se solucionaría con un procesador de mayor velocidad, pero esta solución de fuerza bruta incrementaría tanto el costo como la energía consumida.

Un *benchmark* que ejercite al RTOS en múltiples funciones permitiría al diseñador un razonable asesoramiento sobre el comportamiento del RTOS. Esos programas de evaluación deberían ser independientes del vendedor y contener código consistente para poder comparar los diversos sistemas operativos, es decir que deberán evaluar actividades que estén soportadas por todos los sistemas evaluados.

e) PROCESAMIENTO DE INTERRUPCIONES

Los sistemas en tiempo real son reactivos por naturaleza, respondiendo a eventos externos por medio de interrupciones.

El hardware responde a esos requerimientos transfiriéndole el control a rutinas de atención de interrupción ***(ISRs)*** provistas por el propio RTOS o por el usuario.

El procesamiento del pedido de interrupción provoca que la tarea que se está ejecutando pase al estado "interrumpida" almacenándose su contexto en la pila y se comience a ejecutarse la ISR.

Luego de atender a la misma (que debe ser breve por definición), y dependiendo de la configuración del RTOS, se puede retomar la tarea que se estaba ejecutando previamente o bien pasar a ejecutar la de mayor prioridad que se encuentre "lista".

En cualquier caso, deberá retomar el contexto de la nueva tarea a ejecutar y cargarlo en los registros del procesador.

Esta operatoria que involucra atención de interrupciones, cambios de contexto y transferencia de ejecución será el cuello de botella del RTOS y los dos

parámetros que deberán medirse (y formar parte de los parámetros a ser evaluados[7]) son la latencia de interrupción[1] y la latencia de conmutación de tareas[2].

.

f) SERVICIOS PROVISTOS POR EL SISTEMA

Los RTOS deben planificar y gestionar la ejecución de las tareas o aplicaciones de software.

El RTOS recibe los requerimientos de las tareas para realizar el gerenciamiento, la adjudicación de recursos, el pasaje de mensajes entre las tareas, etc. Estas facilidades que provee el sistema operativo se denominan servicios y deben ser operados por el RTOS muy rápidamente para que la tarea pueda retomarse a la brevedad.

El procesamiento de los servicios del sistema incluye:

- Agendar el inicio de una tarea o hilo luego de la ocurrencia de algún evento futuro.
- Transferir un mensaje de un hilo a otro.
- Solicitar la adjudicación de un recurso de un fondo común.

Cuando hablamos de la evaluación de la actuación del RTOS ante una interrupción, partimos de la base de que la interrupción está bajo el control del usuario, pero en el caso de los servicios de los RTOS éstos son controlador por el proveedor del RTOS y podremos encontrarnos con que un servicio denominado de igual manera en varios RTOS son procesados de distinta forma y con resultados diversos.

La *atención de los servicios* juntamente con la *atención de las interrupciones* son los dos parámetros que caracterizarán la eficiencia del RTOS. Las cifras de calidad provistas por los desarrolladores de los RTOS midiendo estos dos parámetros no son confiables pues en muchas oportunidades han orientado la implementación de los servicios y los resultados de la evaluación en forma tendenciosa a su conveniencia.

g) OTRAS MEDICIONES

Las aplicaciones pueden invocar servicios por dos medios TRAP y CALL.

En la primera, TRAP, (una forma de *software interrupt* actualmente denominada en Cortex SVC = Supervisor Call) se genera un código de operación en el que, los 24 bits menos significativos son ignorados por el procesador[8] y pueden ser empleados por el programador (con la ~~-e~~omplicidad" del compilador) para generar un ~~-ti~~po" de SVC que identifique el tipo de servicio que se quiere invocar.

La rutina de atención de la SVC deberá leer el código de operación y extraer el tipo que se pasó desde la aplicación decodificarlo y actuar según el mismo.

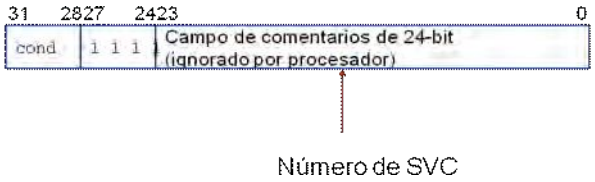

Figura 1: Código de operación de una SVC para un microcontrolador Cortex

Cada diseñador de RTOS tiene la potestad de asignar a cada servicio el tipo de SVC que desee.

La segunda opción CALL, emplea las instrucciones de ramificación (*branch*) del procesador sin interrupciones y si bien sobrecarga menos al procesador, requiere vinculación (*link*ing).

V. SBER. UN SISTEMA BÁSICO DE EVALUACIÓN DE RTOS

La performance de los RTOS es sensible a la plataforma, procesador, reloj, compilador y diseño. Con el objetivo de poder comparar manzanas con manzanas, todos los parámetros anteriores deben ser uniformados para la evaluación entre RTOS.

Presentaremos un conjunto de programas de evaluación de performance, de código abierto e independiente de desarrolladores de RTOS y que se propone evaluar el comportamiento de los RTOS. Este mismo paquete también puede emplearse para comparar procesadores (sobre el mismo RTOS).

Este conjunto de programas buscarán evaluar el comportamiento de varios RTOS tanto en el procesamiento de interrupciones como en el manejo de los servicios básicos comunes en todos los RTOS.

En el contrapunto entre análisis exhaustivo y sólo servicios-básicos optamos por esta segunda opción pues debemos buscar servicios que se encuentren en todos los RTOS habituales.

SBER consiste en un conjunto de *benchmarks* que miden aspectos particulares del comportamiento del RTOS.

- Cambio de contexto en operación cooperativa.
- Cambio de contexto en operación prioritaria ***(preemptive).***
- Procesamiento de interrupciones.
- Procesamiento de interrupciones con preferencia.
- Pasaje de mensajes
- Procesamiento de semáforos
- Asignación y liberación de memoria.

A. Prueba de cambio de contexto en operación cooperativa

Mide el tiempo que tarda el planificador (*scheduler)* del RTOS en cambiar el contexto de un hilo a otro de igual prioridad

- Se crearán cinco hilos de igual prioridad

[1] Tiempo máximo que se tarda en comenzar a atender el pedido de interrupción

[2] Tiempo que transcurre desde que se suspende la tarea en ejecución hasta que se comienza a ejecutar la nueva tarea.

- Cada uno de los cinco hilos se encuentra en un lazo infinito, llamando al servicio de renunciar a la ejecución para delegarla a otro hilo.

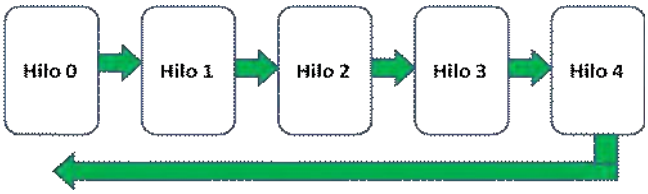

Figura 2: Prueba del cambio de contexto en modo cooperativo

B. *Prueba de cambio de contexto en operación prioritaria (preemptive)*

Mide el tiempo que le toma al RTOS cambiar el contexto de ejecución de un hilo a otro de mayor prioridad que el vigente.

Figura 3: Prueba del cambio de contexto en modo prioritario

- Se crean cinco tareas con prioridades fijas y únicas.
- Los hilos se ejecutan hasta que son suspendidos prioritariamente por un hilo de mayor prioridad.
- Todos los hilos pasan a un estado de suspensión salvo el de menor prioridad.
- El hilo de menor prioridad es sucedido por el de mayor prioridad inmediata, y así sucesivamente hasta llegar al de máxima prioridad.
- Cada hilo incrementa su contador y luego se suspende.
  - Una vez que se completan las suspensiones y el procesamiento regresa al hilo de menor prioridad, el mismo incrementa su contador y se reinicia el proceso.

C. *Prueba del procesamiento de interrupciones*

Se mide el tiempo combinado en iniciar la ISR (interrupciones de *timer* deshabilitadas) + el tiempo necesario para ejecutarla + el tiempo para pasar a través del planificador (*scheduler*) y determinar cual hilo deberá ejecutarse y habilitar la ejecución de dicho hilo, restaurando su contexto si fuera necesario.

En esta prueba, los hilos utilizan interrupciones de software para disparar la priorización y el hilo activado envía resultados cada 30 segundos.

- **Prueba de Interrupciones sin priorización**:

  En este caso el hilo interrumpido es reiniciado sin atender eventuales hilos de mayor prioridad. El tiempo medido es la suma del tiempo de interrupciones inhabilitadas + el tiempo de ejecución de la ISR + el tiempo de actuación del scheduler.

  Debe agregarse la ~~l~~latencia del hilo" a la universalmente aceptada latencia de interrupción.

  Operación:

  1. Se crea una interrupción cuando el hilo 5 genera una TRAP (SVC en Cortex).
  2. El controlador ***(handler)*** de interrupciones se ejecuta y retorna la ejecución al hilo 5 (sistema no prioritario) sin cambio de contexto.

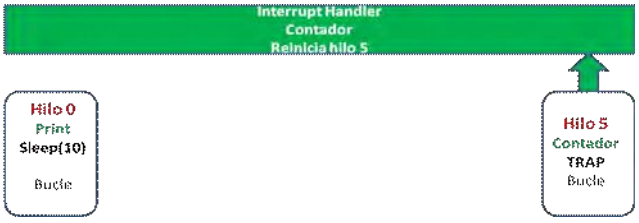

Figura 4: Prueba de la atención de interrupción sin priorización

- En esta prueba consideramos ambas latencias.
- Mediremos cuanto tiempo están deshabilitadas las interrupciones.
- Mediremos cuan rápidamente el hilo de máxima prioridad (el interrumpido) puede ser reactivado.

- **Prueba de Interrupciones con priorización**

Mide el tiempo que se tarda cuando un nuevo hilo se ejecuta luego de la interrupción, en lugar de volver al hilo interrumpido.

Esta situación se presenta cuando, por ejemplo, la interrupción coloca en el estado ~~l~~isto para ejecutar" a un hilo de mayor prioridad que el de la interrupción. De esta manera en la prueba se agrega el tiempo del cambio de contexto al salvar el estado de la interrupción y restaurar el contexto del hilo incorporado.

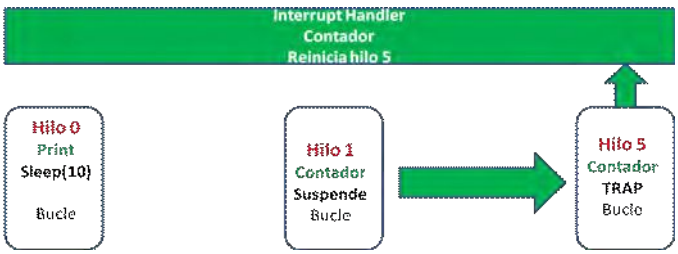

Figura 5: Prueba de la atención de interrupción con priorización

D. *Prueba del pasaje de mensajes*

La forma más común de comunicación entre los hilos es a través de colas.

Ello puede hacerse o bien enviando a la cola el mensaje (pasaje por valor) o bien referenciándolo a través de un puntero y enviando el mismo a la cola (pasaje por referencia).

Realizaremos esta prueba por medio de un mensaje de 16 bytes que referenciaremos por valor.

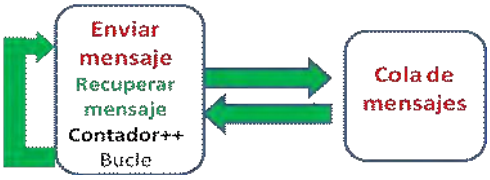

Figura 6: Prueba del pasaje de mensajes por cola

Operación:

1. Un hilo envía 16 bytes a una cola y recupera el mismo mensaje.
2. Una vez que se completó el ciclo se incrementa el contador.
3. Se reitera el bucle en forma perpetua.

E. Prueba del procesamiento con semáforos.

Mide el tiempo que se tarda en tomar y liberar un semáforo binario.

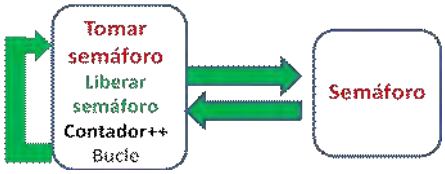

Figura 7: Prueba de la toma y liberación de un semáforo binario

Operación:

1. Una tarea toma ***(get)*** un semáforo binario e inmediatamente lo libera ***(release).***
2. Luego del ciclo toma/liberación el hilo incrementa su contador de ejecuciones.

F. Asignación y liberación de memoria

Se mide el tiempo que le toma a un hilo en el RTOS asignar ***(allocate)*** y liberar ***(deallocate)*** bloques de 128 bytes de memoria.

Algunos RTOS no tienen disponible el manejo de bloques de memoria fijos, en estos casos deberá emplearse un servicio del tipo ***malloc*** que seguramente resultará más lento.

Figura 8: Prueba de la asignación y liberación de un bloque fijo de memoria

Operación:

1. Un hilo asigna un bloque de 128 bytes de memoria e inmediatamente lo libera
2. Luego del ciclo asignación/liberación el hilo incrementa su contador de ejecuciones.

VI. RESULTADOS

Algunos de los sistemas operativos en tiempo real más populares son:

XMOS
eCos
FreeRTOS
Micrium
Xenomai
embOS
ChibiOS/RT
CTL
Nut/OS
TNKernel
ThreadX

Muchos[3] de los cuales prohíben expresamente aparecer en *benchmarks* y hacer referencia a comparaciones con otros RTOS.

ThreadX nos ha dado permiso a publicar los resultados que indicamos en el siguiente cuadro.

Prueba	Valor del contador en 30 segundos
Cambio de contexto cooperativo	1.237.882
Cambio de contexto en operación prioritaria	487.470
Procesamiento de mensajes	830.196
Procesamiento de semáforos	156.6675
Asignación de memoria	140.4046
Manejo de interrupciones no prioritarias	745.664
Manejo de interrupciones prioritarias	316.092

En algunos de los ítems evaluados, en nuestra experiencia encontramos dispersiones de valores superiores al 50%, por lo que recomendamos a los desarrolladores hacer su propia experiencia sobre su hardware y sacar conclusiones definitivas.

Las rutinas de prueba se pueden descargar de ***https://sites.google.com/a/comunidad.ub.edu.ar/rutinas-de-evaluacion-de-rtos/***

VII. MEDICIÓN DEL TIEMPO DE CAMBIO DE CONTEXTO EMPLEANDO UN OSCILOSCOPIO.

Un simple método de medir el tiempo del cambio de contexto es ofrecido por Segger[9] [7] y que consiste en encender y apagar un led (en realidad conmutar el estado de una pata del procesador) y medir el tiempo entre un cambio y el otro por medio de un osciloscopio.

[3] Por ejemplo [1] en pag 177 dice: *"FreeRTOS may not be used for any competitive or comparative purpose, including the publication of any form of run time or compile time metric, without the express permission of Real Time Engineers Ltd. (this is the norm within the industry and is intended to ensure information accuracy)."*

El led se enciende con una señal en la pata del microcontrolador y es apagado por un hilo de mayor prioridad.

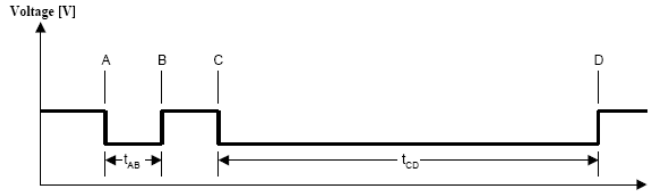

Figura 9: Medición del tiempo de cambio de contexto con un osciloscopio

En la Figura 9, se representa el tiempo t_{CD} que comienza con el encendido del led en C y su apagado desde el hilo en D.

En realidad este tiempo incluye el tiempo propio de encendido del led, por lo que para ajustar la medición se conmuta el led por programa para descontarlo del tiempo mencionado anteriormente. Ese tiempo de calibración es t_{AB}.

El tiempo de cambio de contexto se calculará como:

$$T_{CC} = t_{cd} - t_{ab}$$

VIII. BIBLIOGRAFÍA

La información bibliográfica de la que no se presenta ISBN corresponde a información digital obtenible en internet.

[1] Using the FreeRTOS Real Time Kernel – Richard Barry

[2] Measure your RTOS's real-time performance – William Lamie and John Carbone (2007) Eetimes.

[3] RTCA DO-178B/EUROCAE ED-12B

[4] RTCA DO-178B/EUROCAE ED-12B - Software Considerations in Airborne Systems and Equipment Certification

[5] IEC 62304 - Medical device software – Software life cycle processes

[6] IEC 61508 – Fuctional Safety

[7] Yagarto RTOS comparison

[8] RealView Compilation Tools Developer GuideVersion 4.0.

[9] Segger: Context switching time.

[10] A survey of Real Time Operating Systems for Embedded Systems Development in Automobiles. R. Vamshi Krishna Roll No: 04305015

[11] Real-Time Systems Development Rob Williams Elsevier ISBN-13: 978-0750664714

[12] Real-Time Concepts for Embedded Systems Qing Li, Caroline Yao CMP Books ISBN-13: 978-1578201242

[13] Simple Real-time Operating System: Chowdary Venkateswara Penumuchu Trafford Pub ISBN-13: 978-1425117825