Í N D I C E

CAPITULO 1. Introducción

1.1 OBJETIVOS Y DESCRIPCION DEL PROYECTO

El objetivo principal de este proyecto es el análisis temporal del kernel de tiempo real MaRTE O.S.

Para hacer posible la realización de este proyecto, se han seguido los siguientes pasos:

- Estudio previo
- Instalación de MaRTE O.S.
- Evaluación técnico-práctica de MaRTE O.S.

Actualmente, MaRTE O.S. viene siendo desarrollado por el Departamento de Electrónica y Computación de la Universidad de Cantabria. Este kernel tiene implementado muchas facilidades de tiempo real que cumplen el estándar Posix, pero no están debidamente documentadas en lo que a tiempos de ejecución se refiere. Esto se debe principalmente a la gran variedad de familias de procesadores PC Intel existentes en el mercado.

En este proyecto, se documentarán las medidas mas representativas necesarias para el análisis de planificabilidad de tareas, por medio de sus tiempos de respuesta, en un PC Intel Pentium 120.

1.2 ESTRUCTURA DE LA MEMORIA

Esta memoria está organizada de la siguiente forma:

En el capítulo 2 se realizará un estudio previo de toda la teoría necesaria y así conocer todos los aspectos teóricos a tener en cuenta para el análisis temporal MaRTE O.S.

En el capítulo 3 se realizará una evaluación técnica de MaRTE OS, para conocer todo lo que es posible hacer con este microkernel, y una instalación del entorno de desarrollo que utiliza.

El capítulo 4 es el referente al objetivo principal del proyecto, el análisis temporal de MaRTE O.S.

En el capítulo 5 se realizarán las conclusiones sobre los datos obtenidos y el sistema en sí.

Finalmente, en el capítulo 6 se dará una guía del software y hardware utilizado, y una lista de las palabras claves utilizadas a lo largo del proyecto.

CAPITULO 2. Estudio previo realizado

2.1 DEFINICION DE SISTEMA EN TIEMPO REAL

Para comenzar, habrá que definir la frase "Sistema en Tiempo Real" (S.T.R.) de una forma entendible. Existen muchas interpretaciones de la naturaleza exacta de un STR. Como siempre, todas tienen en común la noción de tiempo de respuesta (el tiempo tomado por un sistema para generar una salida a partir de alguna de las entradas asociadas).

El Diccionario de la Computación de Oxford da la siguiente definición de Sistema de Tiempo Real:

"Cualquier sistema en el cual el tiempo en el que la salida es producida es significativo. En general esto se da porque la entrada se corresponde con algún cambio físico externo, y la salida tiene que indicar ese cambio. El retraso desde el tiempo de entrada al tiempo de salida debería ser suficientemente pequeño para que los 'timeliness' sean aceptables".

Aquí la palabra 'timeliness' es tomada en el contexto del sistema total, es decir, dependiendo para lo que se utilice el sistema, el tiempo de respuesta puede variar entre los milisegundos (ejemplo: un sistema de guía de misiles), o incluso días.

Esta es una definición exacta, aunque se pueden citar otras dos la cuales están enfocadas a los sistemas computacionales:

Young en 1982, define que un STR debe ser:

"cualquier actividad de procesamiento de información o sistema el cual tiene que responder a varios estímulos de entradas externas dentro de un periodo finito y especificado".

The Predictably Dependable Computer Systems da la siguiente definición:

"Un STR es un sistema que es requerido para reaccionar a estímulos generados desde el entorno (incluyendo el paso del tiempo físico) dentro de intervalos de tiempo definidos por el entorno en el que se encuentre."

Como se ve, las definiciones anteriores cubren un amplio rango de actividades de cómputo. Un ejemplo puede ser el sistema operativo UNIX, el cual podría ser considerado un STR cuando un comando es introducido por un usuario y este espera una respuesta en un corto intervalo de tiempo (segundos). Los cajeros automáticos también deberían tener en cuenta esto. Todo esto desemboca en una jerarquía en los STR los cuales pueden ser divididos en:

- HARD REAL TIME
- SOFT REAL TIME

En los sistemas HARD, ocurre que la respuesta tiene que darse obligatoriamente antes de un límite de tiempo (deadline) dado. Por el contrario, los sistemas SOFT son aquellos donde los tiempos de respuesta son importantes, pero el sistema seguirá funcionando correctamente si los deadlines impuestos no se cumplen siempre.

Como es de esperar, las herramientas de desarrollo del software de un STR (lenguajes de propósito general o sistemas operativos utilizados) deberán cumplir con algunos requisitos mínimos para que sea un verdadero STR. Por este motivo se ideó el estándar POSIX1.1003.b el cual intenta unificar y estandarizar los requisitos para los S.T.R..

Para hacernos una idea de dichos requisitos, se pueden citar algunas de las características que deben soportar dichos sistemas:

- Tamaño y complejidad
- Manipulación de números reales
- Extremadamente seguro y satisfactorio
- Control concurrente de los sistemas independientes
- Facilidades de tiempo real
- Interacción con interfaces HW
- Implementación eficiente y entorno de ejecución

2.2 DEFINICION Y CARACTERÍSTICAS DE LOS SISTEMAS EMPOTRADOS

Los sistemas empotrados son sistemas informáticos dedicados para aplicaciones de control. Se podría decir que todos los aparatos electrónicos que incluyan microprocesadores o dispositivos electrónicos de control, son basados en este tipo de sistemas. Pero normalmente suelen carecer de un sistema operativo, lo que les hace ser muy rígidos. Lo ideal sería poder instalarles un S.O. que permitiera que fuesen más configurables e interconectables entre sí, para poder así realizar diversos tipos de tareas adicionales.

Actualmente, el avance de la tecnología permite que lo que pudiera ser un sistema empotrado formado por un sistema software poco adaptable, cerrado y no reutilizable (como por ejemplo un microcontrolador), capaz de gestionar una simple tarea, se convierta por ejemplo en un sistema de apenas unos centímetros cuadrados, alimentado a pilas, con un puerto serie (RS232), memoria RAM y FLASH, un sistema de red ethernet completamente funcional y todo ello controlado por un sistema operativo.

Esta es la filosofía de trabajo y desarrollo que siguen los S.O.T.R. VxWorks, MaRTE OS y uClinux. Los dos primeros tienen un entorno para el desarrollo en base a un PC llamado HOST el cual realiza las tareas de depuración y desarrollo sobre el sistema que es denominado TARGET.

Los sistemas empotrados se caracterizan por los siguientes aspectos:

- Los recursos suelen estar limitados: procesador, memoria, pantalla, etc.
- Los dispositivos de E/S son especiales para cada sistema: no disponen de teclado ni pantalla. El computador debe reaccionar a tiempo ante los cambios en el sistema físico, debido a que una acción retrasada puede ser inútil o peligrosa.

2.3 LENGUAJES Y SISTEMAS OPERATIVOS DE TIEMPO REAL

1.- Lenguajes de Programación

Un lenguaje de programación para sistemas en tiempo real debe facilitar la ejecución de aplicaciones concurrentes, de forma fiable y con un comportamiento temporal analizable y predecible.

Existen varias clases de lenguajes de interés para crear Software para Tiempo Real:

– Lenguajes ensambladores: flexibles y eficientes pero costosos y poco fiables.

– Lenguajes secuenciales (Fortran, Pascal, C, C++): necesitan un S.O. para concurrencia y tiempo real.

– Lenguajes concurrentes (Modula, Ada, Java, etc): concurrencia y tiempo real incluidos en el lenguaje.

Lenguaje C

Es un lenguaje muy utilizado para programación de sistemas, sus principales características son:

– Estructurado, con bloques.

– Sin tipado fuerte.

– Muy flexible (pero a veces poco seguro).

– No tiene integrada la concurrencia ni el tiempo real, esta se consigue invocando servicios del sistema operativo de forma explícita, es decir, por medio de las System Calls.

– No facilita la descomposición en módulos ni la programación con objetos, para ello se creó C++, que es una extensión de C para la programación orientada a objetos, pero no suele usarse en STR por problemas de fiabilidad.

Lenguaje Ada

Es un lenguaje diseñado específicamente para sistemas de tiempo real empotrados, por lo tanto está provisto de:

– concurrencia

– tiempo real

– acceso al hardware e interrupciones

Al ser un lenguaje descendiente de Pascal, hereda lo siguientes aspectos:

– estructura en bloques

– fuertemente tipado

Por último, al estar pensado para construir sistemas grandes y cambiantes entonces nos proporciona las siguientes facilidades de programación:

– paquetes (módulos) y esquemas genéricos.

– extensión de tipos con herencia.

– biblioteca jerárquica.

– interfaces normalizadas con otros lenguajes (C, Fortran).

Lenguaje Ada 95

Es la versión actual normalizada de Ada la cual define:

– un núcleo común para todas las implementaciones (core language)

– unos *anexos* especializados para:

programación de sistemas

sistemas de tiempo real

sistemas distribuidos

sistemas de información

cálculo numérico

fiabilidad y seguridad

Los anexos definen una serie de paquetes de biblioteca y mecanismos de implementación, pero no añaden nueva sintaxis al lenguaje.

Lenguaje Java

Es un lenguaje pensado para construir sistemas distribuidos, por lo que nos proporciona las siguientes cualidades:

– basado en objetos dinámicos

– concurrencia integrada en el lenguaje

– bibliotecas de clases (APIs) muy útiles

– pensado para que el código objeto sea portable, interpretado por la máquina virtual (JVM), cuyo objetivo es: "escribir un programa y ejecutarlo en cualquier plataforma".

Como la definición original no fue pensada para la programación de tiempo real, entonces ocurre lo siguiente:

– la planificación de actividades concurrentes no está bien definida.

– los mecanismos de sincronización son inadecuados.

– la gestión dinámica de memoria introduce indeterminismo.

– la medida del tiempo no es suficientemente precisa

– otros problemas con excepciones y concurrencia

Java para tiempo real

Hay varias propuestas de modificaciones para usar Java en sistemas de tiempo real:

– NIST Requirements for Real-Time Extensions to Java (1999): no modifica la sintaxis, coexistencia con aplicaciones convencionales.

– Java Real-time Experts Group (Sun & otros):Real-Time Specification for Java (2000),basada en un máquina virtual extendida para STR.

– Real-Time Java Working Group (J-Consortium): Real- Time Core Specification basada en una máquina virtual separada para STR.

Nota.- Aun no existen compiladores ni máquinas virtuales para Java de tiempo real.

2.- Sistemas Operativos de Tiempo Real (S.O.T.R)

Los sistemas operativos convencionales no son adecuados para realizar sistemas de tiempo real debido a que:

– no tienen un comportamiento determinista.

– no permiten garantizar los tiempos de respuesta.

– algunos de ellos son poco fiables.

Un sistema operativo de tiempo real (SOTR) debe soportar:

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

– temporización: medida de tiempos y ejecución periódica.

– planificación determinista

ej.: prioridades fijas con desalojo

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

Algunos ejemplos de S.O.T.R existentes en el mercado son:

- Lynx OS
- QNX
- VRTX
- VxWorks 5.3.1
- RT_Linux
- MaRTE O.S

2.4 NORMAS POSIX

1.- Introducción

POSIX es el acrónimo de Sistema Portable para Sistemas Operativos basados en UNIX. Es un estándar que define las interfaces para los servicios que deben proporcionar los programas de aplicación de un sistema operativo para que sea estándar. Es decir, no contiene los detalles de implementación de dichas aplicaciones, si no que las aplicaciones son desarrolladas a gusto del programador. Gracias a esta filosofía, se consigue la portabilidad de las aplicaciones a nivel de código fuente, es decir, que sea posible portar una aplicación de un computador a otro sin más que recompilar su código.

POSIX viene siendo desarrollado por la Computer Society de IEEE, con la referencia IEEE-P1003. Además está siendo estandarizado a nivel internacional con la referencia ISO/IEC-9945.

POSIX está formado por diferentes estándares, los cuales cubren diferentes aspectos de los sistemas operativos. Éstos pueden ser divididos es tres categorías:

Estándares Base

Definen interfases del sistema relacionadas con diferentes aspectos del sistema operativo. Estos estándares especifican la sintaxis y la semántica de servicios para los sistemas operativos, de modo que los programas de aplicación puedan invocarlos directamente.

POSIX.1,1a Interfases del sistema (estándar básico)

POSIX.1b,1d,1j Interfases para la creación y manejo de timers y clocks del sistema.

POSIX.2	Shell y utilidades
POSIX.3	Métodos para medir la conformidad con POSIX
POSIX.4	Extensiones de tiempo real
POSIX.4a	Extensiones de threads
POSIX.4b	Extensiones adicionales de tiempo real
POSIX.6	Extensiones de seguridad
POSIX.7	Administración del sistema
POSIX.8	Acceso a ficheros transparente a la red
POSIX.12	Interfases de red independientes del protocolo
POSIX.15	Extensiones de colas batch
POSIX.17	Servicios de directorios
P1244	Servicios de mensajería electrónica X.400
P1244.1	Interfase para la portabilidad de aplicaciones X.400
P1238	Interfase de comunicaciones O.S.I.
P1238.1	Interfase O.S.I. para la transferencia de ficheros
P1201.1	Interfase gráfica de usuario (ventanas)

Estándares sobre Interfases de los lenguajes de programación

Son estándares secundarios que traducen a un lenguaje de programación concreto los estándares base de los sistemas operativos. Esto permite que puedan ser ejecutadas sobre un sistema operativo dado, cualquier aplicación escrita en un lenguaje definido. Hasta ahora se han desarrollado para los siguientes lenguajes:

POSIX.5	Interfases Ada
POSIX.9	Interfases Fortran 77
POSIX.19	Interfases Fortran 90
POSIX.20	Interfases Ada para las extensiones de Tiempo Real.

Además existe un estándar para el lenguaje C, que es el lenguaje utilizado para la especificación de los estándares base.

Estándares para el entorno de las aplicaciones

Estos estándares incluyen: una guía sobre el entornos POSIX y los perfiles de entornos de aplicación.

Un perfil de aplicación es una lista de los estándares POSIX, con especificación de opciones y parámetros necesarios que se requieren para un entrono de aplicación. El objetivo es conseguir un conjunto pequeño de clases de implementaciones de sistemas operativos bien definidas, y que sean apropiadas para entornos particulares de aplicaciones.

POSIX.0	Guía del entorno POSIX de sistemas abiertos
POSIX.10	Perfil de entorno de aplicaciones de supercomputación
POSIX.11	Perfil de entorno de aplic. de procesado de transacciones
POSIX.13	Perfiles de entornos de aplicaciones de tiempo real
POSIX.14	Perfil de entornos de aplicaciones multiprocesadoras
POSIX.18	Perfil de entornos de aplicación de plataformas POSIX

Una vez comprendido en que consiste POSIX, vamos a centrarnos en todo lo referente a las especificaciones de Tiempo Real.

2.- Estandarización de los S.O.T.R

Las aplicaciones de tiempo real se caracterizan porque, el funcionamiento correcto de las mismas no sólo depende de los resultados del cálculo, sino también del instante en el que se generan estos resultados. Por este motivo, sería deseable que el sistema en el que se trabaje tenga un comportamiento temporal predecible. Esto implica que los servicios utilizados tengan un tiempo de respuesta acotado.

Debido a esto, en POSIX se creó un grupo de trabajo de tiempo real con el objetivo de conseguir la portabilidad de las aplicaciones de tiempo real. Éste grupo se encarga de añadir al POSIX básico los servicios necesarios para desarrollar aplicaciones de tiempo real. Actualmente se han diferenciado entre los perfiles para los lenguajes, y perfiles para los entornos de ejecución.

Perfiles para los lenguajes de tiempo real

POSIX.4 Extensiones de tiempo real: Define interfases para soportar la portabilidad de aplicaciones con requerimientos de tiempo real.

- Planificación de procesos de tiempo real
- Inhibición de la memoria virtual
- Sincronización de procesos
- Memoria Compartida
- Señales de tiempo real
- Comunicación entre procesos
- Relojes y temporizadores
- E/S asíncrona
- E/S sincronizada

POSIX.4ª Extensiones de thread: Define interfases para el soporte de múltiples threads o flujos de control dentro de cada proceso POSIX.

- Control de thredas
- Planificación de threads
- Sincronización de threads

POSIX.4b Extensiones adicionales de tiempo real: Define interfases para el soporte de servicios de tiempo real adicionales tales como:

- Tiempos límite (Timeouts)
- Relojes de tiempo de ejecución
- Servidor esporádico
- Control de interrupciones
- Control de dispositivos de E/S
- Creación de procesos

Perfiles para los sistemas operativos de tiempo real:

POSIX.13 Perfiles para entornos de aplicaciones de tiempo real. Cada perfil especifica una lista de los servicios que se requieren para un entorno de aplicación particular.

Éste último interfaz, facilita mucho las cosas a la hora de elegir un entorno de aplicación, debido a la infinidad de sistemas existentes. Por ejemplo, sea un sistema empotrado, en el que la limitación de recursos físicos es tan importante que pueden no disponer de disco duro, no tener unidad hardware de manejo de memoria (M.M.U.), no disponer de consola de comunicación, etc. Para estos sistemas es necesario que el estándar permita la utilización de un conjunto reducido de funciones o servicios, ya que el resto lo haría más grande e ineficiente. Por lo tanto, a la hora de crearnos un sistema operativo (o entorno de ejecución) para aplicaciones embebidas, habrá que hacerlo siguiendo los siguientes esquemas:

PSE50.- Sistemas empotrados pequeños: Corresponde a un sistema empotrado pequeño sin necesidad de unidad de manejo de memoria, sin sistema de ficheros, y sin terminal de E/S . Solo se permite un proceso, aunque puede haber múltiples threads ejecuntándose de forma concurrente.

PSE51.-Controladores de tiempo real: Corresponde a un sistema controlador de propósito especial. Es como el perfil mínimo, pero añadiendo un sistema de fichero y un terminal de E/S. Solo se permite un proceso, aunque se permiten múltiples threads.

PSE52.-Sistemas empotrados grandes: Corresponde a un sistema empotrado grande, sin sistema de ficheros. Puede tener múltiples procesos y múltiple hebras.

PSE53.-Sistemas grandes con requerimientos de tiempo real: Corresponde a un sistema grande de tiempo real que soporta todos los servicios.

Una vez entendida la necesidad de la estandarización de los sistemas y teniendo en mente todo lo relacionado con la programación en tiempo real, será fácil comprender la potencia del microkernel que se va a ser analizado en este proyecto. MaRTE OS fue inicialmente diseñado para ceñirse por completo al perfil PSE50, y todos los perfiles relacionados con el soporte de aplicaciones en tiempo real para sistemas empotrados. Por lo tanto, es un entorno de ejecución adecuado para la ejecución de aplicaciones de tiempo real sobre sistemas empotrados con recursos limitados.

2.5 ESTUDIO DE VxWORKS 5.3.1

1.- Introducción

VxWork 5.3.1 es un RTOS que viene siendo desarrollado por WindRiver Systems Inc. y está basado en la filosofía HOST-TARGET para sistemas empotrados.

VXWorks era inicialmente un entorno de red y un desarrollador de aplicaciones para VRTX y pSOS. Después Wind River Systems desarrolló su própio microkernel. El resultado es que VxWorks a evolucionado hasta convertirse en una herramienta de cliente-servidor.

VxWorks 3.5.1 se caracteriza por tener:

- Buen entorno de desarrollo con un amplio conjunto de herramientas.
- Extenso interfaz para aplicaciones
- Ejecución Predecible
- Soporte técnico profesional.
- Soporta muchas plataformas HW diferentes.

Aunque no está bien preparado para:

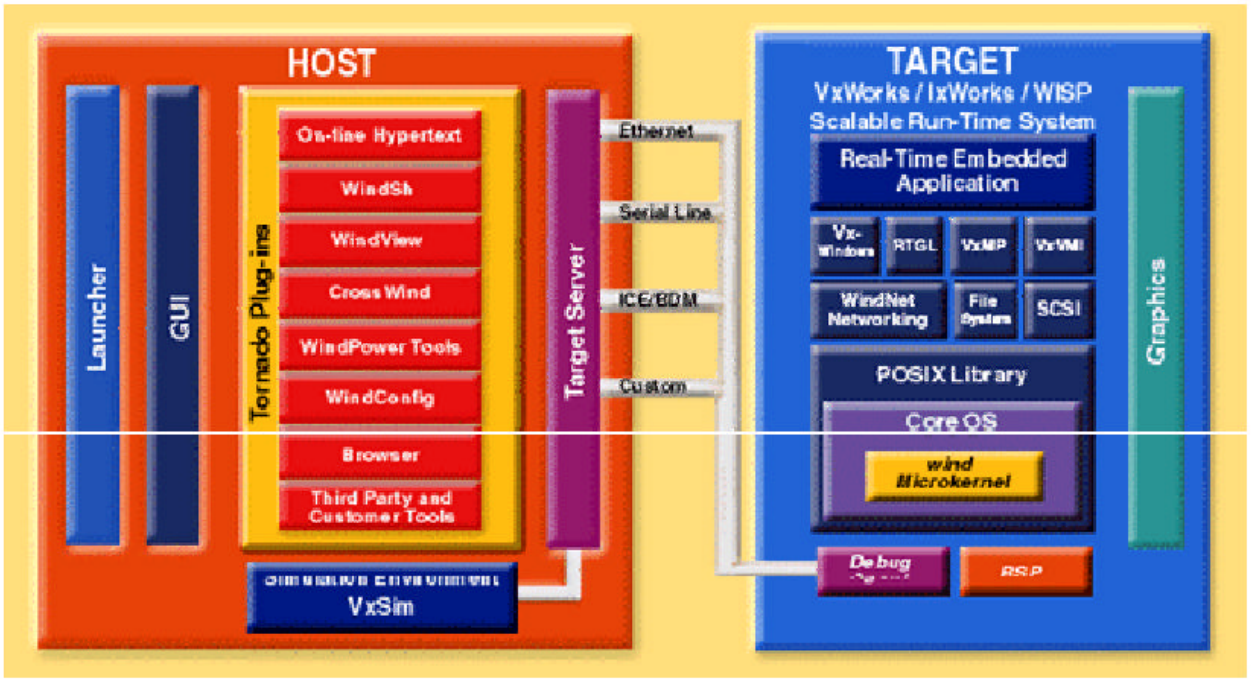
- Tolerancia a fallos
- Sistemas distribuidos
- Soporte pobre para la comunicación entre distintos procesadores.
- Los manuales no son muy claros.

2.- Evaluación técnica de VxWorks

Entorno de desarrollo

El RTOS VxWorks 5.3.1 se ejecuta en la tarjeta, mientras que Tornado 1.0.1 (que es el entorno de desarrollo de las aplicaciones, se ejecuta en el HOST). Ambos sistemas están conectados por medio de una red Ethernet u otro tipo de comunicación.

Siguiendo esta filosofía, Wind River utiliza dos S.O. complementarios, y deja que cada uno haga lo mejor. Por un lado corre un S.O. de propósito general como Windows NT o UNIX, que es utilizado solo para aplicaciones de desarrollo. En la tarjeta, en la cual corre VxWorks, es donde se manejan las tareas de Tiempo Real. Todas las herramientas SW se ejecutan en el HOST, mientras que en el TARGET se encuentra únicamente un agente para la depuración remota de las tareas.



El entorno de desarrollo es completo para HOSTS basados en UNIX y Windows.

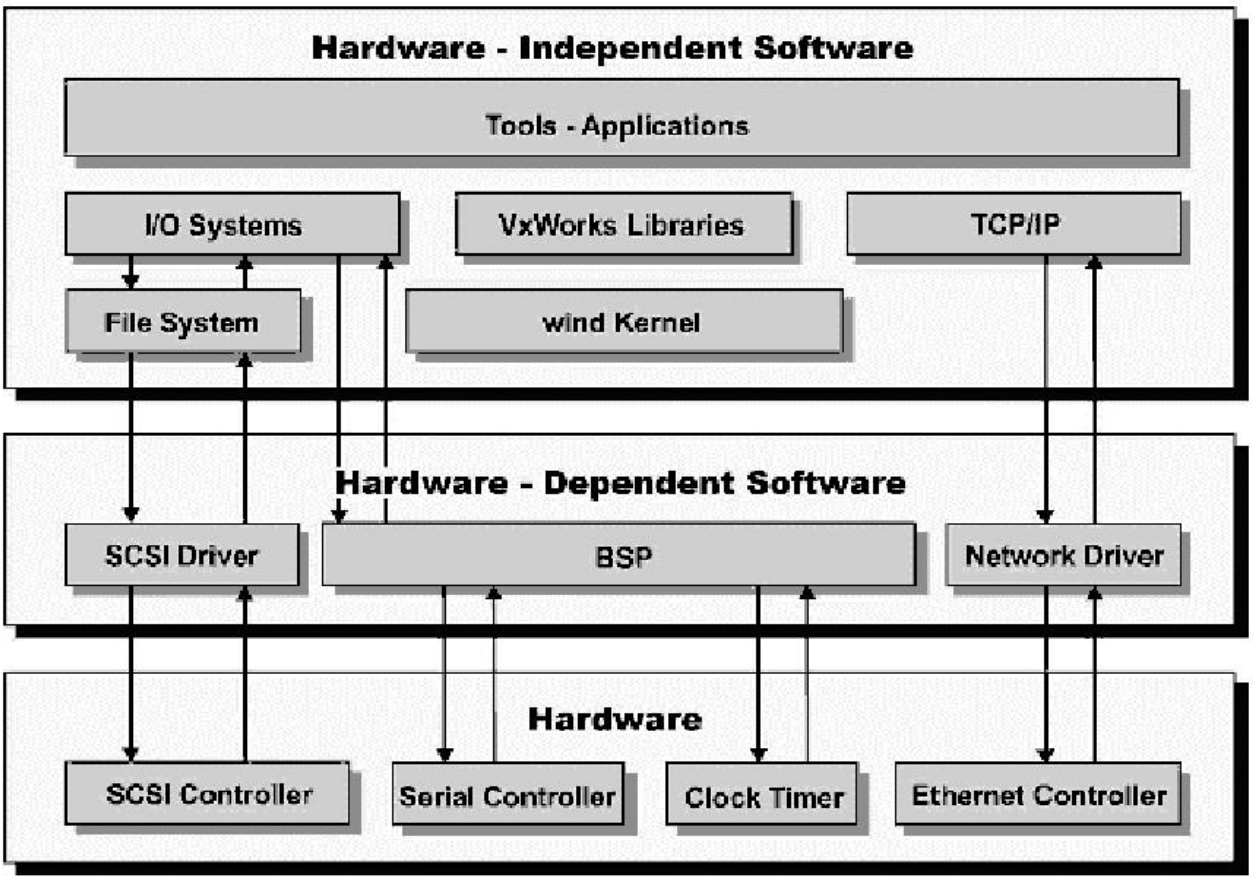
Para asistir a los desarrolladores de sistemas embebidos usando los HW habituales, el sistema WindRiver también ofrece VxSim, que es una herramienta de prototipado y simulación para Tornado/VxWorks. VxSim proporciona una simulación de VxWorks en el

HOST. Con esta herramienta, el desarrollo de aplicaciones puede empezar antes de que el HW llegue a estar disponible.

Arquitectura del kernel

VxWorks consistía inicialmente en un entorno de red y un desarrollador para VRTX y pSOS. Después Wind River Systems desarrolló su propio microkernel. Como resultado se obtiene que VxWorks es una herramienta basada en las arquitectura cliente-servidor.

En el corazón del entorno de ejecución, se encuentra el microkernel de wind. Este microkernel soporta un alto rango de características de tiempo-real incluyendo multi-tarea, scheduling, comunicación y sincronización entre trareas, y manejadores de memoria. El resto de funciones son implementadas como procesos.



Nota.- VxWorks no utiliza privilegios de protección. El nivel de privilegio es siempre el 0 (modo supervisor), por lo tanto no es necesario diferenciar entre llamadas al sistema y de usuario, ya que comparten los mismos privilegios.

Nota.- VxWord tiene una comunicación pobre entre procesos localizados en diferentes procesadores. Por ejemplo, las colas de mensajes solo pueden ser usadas con memoria compartida. Además debería de instalarse el componente VxMP para poder tener zonas de memoria independientes.

Sistema básico facilitado

VxWorks da soporte a las siguientes primitivas de programación:

- Método de manejo de tareas (entorno básico de multitarea), proporcionando planificación con prioridad preemptiva o Round Robin.
- Método de manejo de memoria, utiliza mismo espacio de memoria para todas las tareas y el kernel.
- Método de manejo de interrupciones.
- API Posix. VxWorks 5.3.1 también soporta compatibilidad con el estándar POSIX 1.003.b.

3.- Herramientas

El entorno de desarrollo de Tornado interactúa con un conjunto rico de herramientas SW. Además, Tornado es un entorno abierto completo, haciendo fácil este para integrar herramientas desarrolladas desde los usuarios o desarrolladores.

Tipos de herramientas disponibles actualmente:

- Editores
- Compiladores
- Linkadores
- Herramientas de desarrollo
- Depuradores
- Herramientas para el análisis del sistema
- Cargadores

4.- Resumen

VxWorks da respuestas temporales del orden de los nanosegundos. Por lo tanto es un sistema adecuado para la planificación de tareas que tengan requerimientos temporales muy exigentes.

VxWorks está disponible para una gran variedad de plataformas SW como Windows-NT, Unix, etc.

VxWorks no trabaja bajo la licencia GNU, por lo tanto, hay que pagar por una licencia de este sistema operativo.

CAPITULO 3. Instalación y evaluación técnica de MaRTEO.S.

3.1 DEFINICION DE MaRTE OS

MaRTE OS (Minimal Real-Time Operating System for Embebed Applications) es un kernel de tiempo real para aplicaciones embebidas el cual sigue el subconjunto del perfil POSIX.13 para Sistemas de Tiempo Real Mínimo (PS5.0), proporcionando los interfaces POSIX para los lenguajes C y ADA.

Este subconjunto requiere solo un conjunto reducido de servicios del sistema, los cuales pueden ser implementados como un kernel pequeño y muy eficiente que puede ser usado para implementar sistemas empotrados con requerimientos de tiempo real.

3.2 INSTALACION DE MaRTE OS

3.2.1 Instalación del entorno de trabajo

El entorno de trabajo y desarrollo que será instalado, es el compuesto por los siguientes elementos SW y HW:

HOST

- AMD K6-2 500 MHz
- Linux Red Hat 6.0
- Gnat 3.13
- Gcc 2.8.1
- Bootp-2.4
- Tarjeta Ethernet Real Teck
- Monitor
- Disco duro
- Memoria RAM

TARJETA

- Intel Pentium 120
- Disquetera 1.44 Mbytes
- Tarjeta Ethernet Real Teck
- Memoria RAM

1.- Instalación del Sistema Operativo Linux Red Hat 6.2

Esta es la distribución donde se ha instalado y testeado MaRTE OS para su correcto funcionamiento.

Antes de realizar la instalación de cualquier distribución, habrá que disponer la información sobre las características del equipo en el cual se va ha instalar:

Nota.- al menos hay que tener un par de disquetes formateados para crear los discos de rescate y arranque.

La información más relevante del sistema puede ser obtenida cumplimentando la siguiente ficha:

Numero de discos duros:	1
Tamaño de cada disco duro:	13 GBytes
Disco duro principal:	HDA
Cantidad de memoria RAM:	128 MBytes
Tipo y número de unidades de CD-ROM:	1 IDE
Marca y modelo de cada CD-ROM:	PIONER X24
Marca y modelo de adaptadores SCSI:	---
Tipo de ratón:	PS/2
Número de botones del ratón:	3 Botones
Si es serie, puerto COM al que está conectado:	---
Marca, modelo, y memoria RAM de la tarjeta gráfica:	VOODO BANSHEE 16 MBytes
Marca y modelo del monitor:	PHILIPS 14 C
Tasa de refresco horizontal permitida:	55-70
Tasa de refresco vertical permitida:	---
Red:	ETHERNET
IP:	192.168.1.0
Máscara de red:	255.255.255.0
Dirección de salida:	192.168.1.255

Dirección IP del nombre del dominio del servidor: ---

Nombre del dominio: linux.mio

Nombre del host: comphostX

Marca y modelo de la tarjeta de red: REAL TECK

Sistemas operativos adicionales instalados: WINDOWS 98

Ubicación de dónde irá instalado el LILO (si se va a utilizar): HDA

Tener instalado Master Boot Record (necesario para la utilización de LILO): SI

Partición de Linux instalada con anterioridad (si se quiere utilizar LILO): ---

Una vez cumplimentada toda esta información, se podrá comenzar con la instalación. La forma más fácil es arrancar el PC con el CD de Linux Red Hat 6.0 puesto en el CDROM, previamente habiendo cambiado los parámetros de arranque de la BIOS del PC para que arranque con dicho CD.

Una vez arrancado, tecleamos Expert en el prompt del arranque y seguiremos los pasos de la instalación los cuales son guiados por medio de un entorno gráfico amigable.

Nota.- Durante este proceso de instalación, tendremos que utilizar los parámetros obtenidos en la ficha anterior.

En lo que sigue, se explicará solamente lo que está más relacionado con la instalación de MaRTE O.S.

2.1 Creación de las Particiones del Disco Duro

En la instalación actual, se tomará la siguiente estructura para el disco duro:

Sea Hdda el primer HDD conectado como el maestro del primer conector IDE de nustro PC, entonces la asignación de las particiones será:

hda1	Windows	
hda2	Linux	/home
hda3	Ampliada	
hda5	Linux	/
hda6	Swap	

Nota.- Es muy aconsejable crear una partición '/home' y otra '/' bien diferenciadas, por si en un futuro hay que formatear alguna de ellas y así no perjudicar a la otra.

Nota.- En dicha partición se ha reservado una pequeña zona para el Sistema Operativo Windows porque me he encontrado aplicaciones a utilizar en el proyecto que solo estaban disponibles para dicha plataforma.

3.- Creación del entorno de Red

La configuración de la red local para el entorno de desarrollo Host-Target de las aplicaciones, es el siguiente:

Red:	Linux.mio	192.168.1.0
Host:	CompHostX	192.168.1.X0
Target:	CompMarteY	192.168.1.XY

Donde X e Y nos permitirán poder direcciones independiente de red para diferentes instalaciones del entorno de desarrollo de MaRTE OS.

4.- Instalación de MaRTE OS

Antes de instalar MaRTE OS, tendremos que poseer en nuestro sistema los siguientes paquetes:

gnat-3.13p-i686-pc-linux-gnu-bin.tar.gz: Es el compilador de Gnat para Linux, que será necesario a la hora de la instalación de Marte OS.

marte-1.0.tar.gz: Es el microkernel MaRTE OS con todas las utilidades de instalación, librerías y manuales.

bootpd-2.4.tar.gz: Es el protocolo con el cual se ha realizado el arranque remoto de MaRTE OS desde el HOST.

Nota.- Al ser todos paquetes fuentes, necesitaremos el compilador gcc y el make instalados en el sistema.

Una vez que tenemos estos paquetes en el sistema, los instalamos:

#tar -zxfv gnat-3.13p-i686

#cd gnat-3.13p

#./doconfig

en este paso, se indicará donde queremos que se instale gnat, para ello ejecutamos el programa de instalación y añadimos la ruta elegida en el path (en este caso ha sido /usr/gnat/bin).

#./doinstall

#PATH=/usr/gnat/bin:$PATH

Una vez realizados estos pasos, ya tenemos instalado el compilador Gnat para Ada95 en nuestra máquina. Ahora procederemos con la instalación de MaRTE OS:

a) Descomprimimos las fuentes

```
#tar -zxvf marte-1.0.tar.gz
```

Un truco muy útil es renombrar al directorio marte con otro nombre y después, un enlace a ese directorio con el nombre marte, para así poder poder tener varias instalaciones de MaRTE en la máquina:

```
#mv marte marte1.0A
#ln -s marte1.0A marte
```

b) Ejecutamos el script de autoinstalación de MaRTE. Este nos llevará a través de una serie de etapas, en las cuales se nos pedirán una serie de datos útiles para MaRTE O.S.:

```
#cd marte
#./minstall
```

b.1) Tipo de procesador del puesto que hará de TARGET. Esta información le servirá a MaRTE OS para conocer la arquitectura de PC en la que se monta, y así poder utilizar el Time Stamp Counter (el cual es mucho mas eficiente para medir tiempos).

b.2) Seremos preguntados por la versión y el path de instalación del Gnat. Podremos cambiarlo si es necesario.

Una vez realizado este paso, comenzará a instalarse MaRTE O.S. creándose los ficheros, enlaces, directorios, etc, para su correcto funcionamiento.

b.3) Antes de la completa instalación, se nos preguntará por algunos parámetros necesarios para el arranque remoto del entorno de desarrollo:

- La IP del HOST:

  192.168.1.20

- Directorio que será exportado por la TARGETA desde el HOST que contiene las aplicaciones del usuario:

  /home/MarteOS/export/marte

- Nombre de IP de la TARGETA:

  compmarte2.linux.mio

- Tipo del interfaz de red a utilizar:

  ne200

Y una vez introducidos estos datos, el script nos pedirá un disco de 1.44 MB para poder crear un disquete de arranque.

5.- Configuración de MaRTE O.S.

Una vez instalado el microkernel y obtenido el disquete de arranque, procederemos con la configuración de los parámetros y entorno de red para nuestro sistema:

a) Primero añadimos a nuestro path del sistema el directorio /utils, el cual contiene todas los scripts de compilación del kernel y aplicaciones:

```
#export PATH=$PATH:/home/MarteOS/marte/utils
```

y lo exportamos al directorio que será montado desde la targeta. Para ello ejecutamos el script interno

```
#mexport /etc/export
```

el cuál añadirá la siguiente línea al fichero especificado:

/home/MarteOS/export/marte

y activa los cambios ejecutando:

exportfs -a

b) Ahora necesitaremos ver si tenemos instalado el sistema de ficheros en red (NFS, que contiene el protocolo de red para trabajar de forma remota), y permitirá a MaRTE O.S. descargarse las aplicaciones del usuario desde el HOST. Si no estuviera instalado, entonces habría que instalarlo.

Consultamos si se está ejecutando (normalmente suele haber varios) el demonio nfsd:

#ps -ex | grep nfsd

si aparecen varias líneas como esta, entonces es que se encuentra ejecutando,

PID ? SW 0:00 [nfsd]

si no, lo activamos:

/etc/rc.d/init.d/nfsd start

6.- Configuración del protocolo de arranque remoto bootp

Una vez llegado a este paso, nos encontramos en el momento mas delicado de todo el proceso (aunque si se hace bien, no tiene que dar ningún problema), la instalación del protocolo de arranque remoto bootp.

El protocolo bootp utilizado para la instalación de MaRTE OS en el completo entorno de desarrollo, es una revisión del original, que sigue el formato definido por el RFC 1048. La forma de configurarlo una vez instalado, es la siguiente:

a) Primero habilitamos los protocolos necesarios para el arranque remoto especificados en /etc/inet.conf. Esto consistirá en decomentar las siguientes líneas:

```
# tftpd  dgram  udp     wait     root     /usr/bin/tcpd     in.tftpd
# bootps         dgram  udp     wait     root     /usr/bin/tcpd     bootpd
```

si estuvieran decomentadas, entonces solamente habría que añadir a bootpd, in.bootpd, es decir:

```
bootps dgram  udp     wait     root     /usr/bin/tcpd in.bootpd
```

b) Habilitamos los servicios en /etc/services, añadiendo o decomentando:

```
bootps          67/tcp          #BOOTP          server
tftp            69/udp          #TFTPD          server
```

c) Ahora instalamos el paquete bootpd-2.4.tar.gz, que es el protocolo en sí:

```
#tar -zxfv  bootpd-2.4.tar.gz
#cd bootpd-2.4
#make linux
#make installa-linux
```

Y no olvidemos de resetear el superdemonio de servicios de red:

#/etc/rc.d/init.d/inet restart

d) Finalmente, una vez corriendo demonio bootpd, lo configuramos por medio de su fichero de configuración /etc/bootptab, que en nuestro caso quedaría de la siguiente forma:

```
.default:/
        :sm=255.255.255.0:/
        :gw=192.168.1.255:/
        :vm=rfc1048:/
        :ht=ethernet:/
        :hd=/home/dani/export22B/marte:/
        :bf=mprogram:/
        :ds=192.168.1.20:/
        :hn:
compmarte22:ha=00c0262455be:ip=192.168.1.22:tc=.default:
```

Una vez realizado todo este proceso, estamos en disposición de poder comenzar a trabajar con MaRTE OS utilizando el entorno de desarrollo básico por medio de una red interna LAN.

Nota.- Cada instalación tendrá los parámetros específicos para su entorno.

3.2.2 Recompilación de MaRTE O.S.

Para el objetivo final de este proyecto (realización de test para el comportamiento temporal de MaRTE O.S), será necesario recompilar el kernel para poder añadirle más tareas, memoria, etc. Para esta tarea, MaRTE OS dispone de una serie de scripts de compilación situados en /utils. los cuales han sido empleados para la instalación del kernel. El propósito y manejo de estos scripts es el siguiente:

1.- Compilar las librerias y kernel de MaRTE O.S

El kernel y las librerias están compilados usando con una completa optimización de las opciones durante la etapa de instalación. Por lo tanto existen algunas razones por las cuales se puede estar interesado en recompilar el microkernel MaRTE OS:

- En el caso que se estén realizando algunas modificaciones en MaRTE OS o añadiendo algunas nuevas funcionalidades.

- Si se desea compilar el kernel con algunos switches de compilación diferentes de los que se realizó por defecto en la instalación.

- Si se desea habilitar algunos chequeos de depuración en el kernel.

- Si se quiere utilizar la herramienta task_inspector para analizar tareas.

- En orden de cambiar el numero máximo de recursos permitidos en una aplicación de MaRTE O.S.

Para ello, existen algunos scripts en el directorio utils/ que te permiten recompilar el kernel y/o las librerias internas:

mkkernel:

Este script compila el kernel, para ello acepta opciones de gnat y/o gcc, así si se desea un kernel optimizado para que sea rápido, habrá que ejecutar:

```
$ mkkernel -gnatn -03 -gnatp
```

El flag -f puede ser añadido para forzar la recompilación de todo el kernel, por el contrario solo los paquetes modificados y relacionados con éste serán compilados.

mklibmc:

Este script genera la versión de la librería standar de C libmc para MaRTE OS, lib/libmc.a. Para una completa optimización de dicha librería habrá que ejecutar

```
$ mklibmc -03
```

mkall:

Hace lo mismo que mkkernel y mklibmc cuando se realizan juntos pero forzando la recompilación de todo, es decir, no solo de los ficheros modificados. Es como volver a instalar MaRTE OS de nuevo pero sin cambiar la arquitectura de la targeta, los datos del host y la versión de gnat. También acepta las opciones de gnat y/o gcc, así si se desea un kernel completamente optimizado habrá que ejecutar:

```
$ mkall -gnatn -03 -gnatp
```

2.- Optimización del tamaño y velocidad del kernel

Por defecto el kernel es optimizado para ser rápido (es decir, con los flags -gnatn -gnatp -03) y sin ningún pragma. Si se quisiera imponer algunas restricciones para las aplicaciones o se desea utilizar una versión simplificada del run-time con el uso del pragma "Restricted_Run_Time" o "Ravenscar" lo único que habrá que hacer es crear un fichero gnat.adc con los pragmas deseados en el directorio de la aplicación y forzar la recompilación de todo con los flags -a y -f.

Por ejemplo, para un tamaño mínimo del kernel, se crea el siguiente fichero gnat.adc con los siguientes pragmas:

```
pragma Ravenscar;
pragma Restrictions (Max_Tasks => 2);
```

y entonces todo (código de la aplicación, kernel de MaRTE OS y la parte de Gnat Run_Time requerida para la aplicación) debería ser recompilada con el comando:

```
$ mgnatmake -gnatp -03 -a -f aplicación.adb
```

3.- Habilitación de los chequeos y mensajes suministrados por el kernel

Si algunos chequeos de depuración son incluidos en el kernel en la forma de pragmas "Assert". Ellos pueden ser usados para detectar errores internos dentro del kernel antes de que ellos causen algún comportamiento inesperado imposible de analizar. Para habilitar estos chequeos el kernel tiene que ser recompilado con el flag '-gnata':

```
$ mkkernel -gnata –f
```

Habilitar los pragmas "Assert" puede ser útil, si se está modificando MaRTE OS para chequear la consistencia de los cambios realizados.

El kernel también puede ser configurado para visualizar por pantalla algunos o todos los mensajes de depuración en la consola. Ellos pueden informarnos de todos los eventos relevantes sobre el cambio de contexto, mutexes, señales, eventos temporizados, etc. Esta funcionalidad puede ser útil para los desarrolladores del kernel en algunas situaciones específicas. En el caso general, el número de mensajes visualizados suele ser demasiado grande para ser analizado, y el uso del depurador podría ser deseado. Para habilitar los mensajes de depuración, habrá que editar el fichero 'kernel/debug_messages.ads', poniendo a true algunos o todos las constantes booleanas debajo de la etiqueta "General Messages" y recompila el kernel con los asertos habilitados:

```
$ mkkernel -gnata -f
```

4.- Cambio de los parámetros de configuración del kernel

MaRTE OS es un sistema estático en el cual el número máximo de recursos (por ejemplo, hebras, semáforos, pila de hebras, número de niveles de prioridad, temporizadores, etc), es definido en el tiempo de compilación. Si se desea cambiar los valores por defecto, el kernel deberá de ser recompilado con estos nuevos valores.

Los parámetros de configuración son definidos como constantes en el fichero:

'kernel/configuration_parametrers.ads'

Debajo de cada parámetro hay una breve descripción de su significado.

Por ejemplo con los valores por defecto, solo 8 tareas en Ada pueden ser creadas, aunque en la constante 'Num_tasks_MX' es 12 por defecto. El problema está relacionado con la cantidad de memoria dinámica usada por Gnat para crear una tarea (unos 12 Kb por tarea). Para superar este límite también habrá que cambiar el tamaño de la cola de memoria dinámica. Para ello se edita el fichero 'kernel/configuration_parameters.ads' y se cambia el valor asignado a la constante 'Dynamic_Memory_Pool_Size_In_Bytes' (100kb por defecto). Si quisiéramos crear más de 12 tareas en nuestras aplicaciones, deberíamos cambiar también la constante 'Num_Tasks_Mx'.

Después de todos estos cambios, tendremos que recompilar el kernel con el comando:

```
$ mkkernel -gnatn -gnatp -03
```

Una vez claro todo esto, se procederá al ajuste del kernel para poder realizar los TESTS que se especificarán en el apartado correspondiente. Las principales necesidades de dichos test son: muchas tareas y aumento del spool de memoria dinámica. Para ello, se recompilará el kernel con los siguientes parámetros pertenecientes al fichero de configuración /kernel/configuration_paramameters.ads:

Num_Tasks_Mx = 30;

Dynamic_Memory_Pool_Size = 400 * 1024;

y se procederá a la recompilación:

#mkkernel -gnatn -gnatp -03

En esto momento, ya tenemos el kernel listo para ser utilizado.

3.2.3 El protocolo de arranque remoto BOOTP

El protocolo BOOTP que se utiliza para el arranque remoto de las aplicaciones. Es una mejora del servidor CMU BOOTP derivado del servidor BOOTP original creado por Hill Croft y Stanford.

El demonio BOOTPD es útil siempre que se utilice en conjunto con el protocolo de ficheros de red NFS. Este protocolo permitirá tener acceso al sistema de ficheros del HOST, y así poder bajar la aplicación a ejecutar en la TARJETA para el arranque remoto. Además también permite que el código sea almacenado en una EPROM, por lo tanto podremos tener una computadora aislada en cualquier lugar, con una conexión de red y sin ningún tipo de interfaz de entrada salida.

Otra alternativa para el arranque remoto sería utilizar el protocolo RARP, pero BOOTP es mucho mas flexible, ya que añade ficheros de configuración para una gran diversidad de clientes.

El formato que siguen los paquetes bajo este protocolo, ha sido realizado siguiendo las especificaciones del RFC1048. Los RFC's son documentos de texto formales, que definen nuevos protocolos de servicios de red (FTP, Telnet,…) o mejoras de los mismos (indicando si se han quedado obsoletos o no). En relación con el bootp existe el RFC951, RFC1048, RFC1499, etc.

1.- Funcionamiento del protocolo bootp

El Bootstrap Protocol (BOOTP) es un protocolo basado en paquetes UDP/IP (es decir, orientado a la conexión), que proporciona un arranque para la máquina host, pudiendo así configurarse ella misma dinámicamente y sin supervisión del usuario. BOOTP proporciona un camino de comunicar a un host cliente una dirección IP asignada, la dirección IP de un host servidor de arranque, y el nombre de un fichero a ser cargado en la memoria y ser ejecutado. Otras informaciones de configuración, tales como la máscara local de la subred, el desplazamiento de tiempo local, las direcciones de los routers por defecto, y las direcciones

de varios servidores de Internet, pueden también ser comunicadas al host usando este protocolo.

2.- Paquetes y flags del protocolo bootp

La especificación de los paquetes que utiliza este protocolo, están definidos por una zona UDP y otra BOOTP. El campo perteneciente al UDP puede ser puesto a cero por el cliente o el servidor para eliminar la sobrecarga en una implementación PROM.

La descripción de los campos es la siguiente:

CAMPO	BYTES	DESCRIPCION
op	1	código del paquete/tipo de Mensaje 1 = BOOTREQUEST, 2 = BOOTREPLY
Htype	1	Tipo del hardware direccionado '1' = 10mb ethernet
Hlen	1	longitud de la dirección hardware (eg '6' for 10mb ethernet).
Hops	1	cliente puesto a cero, opcionalmente usado por los gateways en arranques a través de estos.
Xid	14	ID de transacción, un número aleatorio,utilizado para adjudicar el requerimiento de arranque con las respuestas que generan.
Secs	2	Puesto por el cliente, segundos transcurridos desde que el cliente comenzó el intento de arranque
--	2	No usado.
Ciaddr	4	Dirección IP del cliente; puesta en, y para el cliente en el requerimiento de arranque, si es conocida.
Yaddr	4	La dirección IP del cliente; Puesta por el servidor si el cliente no conoce su própia dirección. (cuando ciaddr está a 0);
Siaddr	4	Dirección IP del servidorr; devuelta en la respuesta de arranque por el servidor.
Giaddr	4	Dirección IP del gateway, usada opcionalmente en un arranque por medio de gateways.

Chaddr	16	Dirección hardware para el cliente, puesta en y para el cliente.
Sname	64	Nombre opcional del host servidor. Puede ser la cadena vacia
File	128	Nombre del fichero de arranque, puede ser nulo. Y path completo en la respuesta del arranque.
Vend	64	Area opcional específica para el vendor.

3.- Comportamiento del cliente

El flag BROADCAST

Normalmente, los servidores BOOTP y los agentes intermedios, intentan mandar mensajes BOOTREPLY directamente al cliente usando mensajes unicast. La dirección IP de destino (en la cabecera de IP), es activada en la dirección BOOTP 'yiaddr'. La dirección de nivel de enlace, es activada en la BOOTP 'chaddr'.

Desafortunadamente, algunas implementaciones de los clientes no están permitidas para recibir tales datagramas IP, hasta que estos no conocen sus propias direcciones IP. Sin embargo, ellos pueden recibir datagramas IP por el broadcast.

Si un cliente cae en esta categoría, debería poner a 1 el flag de broadcast en el campo de BOOTREPLY del mensaje que genera. Esto proporcionará una ruta indirecta a los servidores BOOTP y agentes relay, que deberían intentar direccionar sus mensajes BOOTREPLY al cliente.

Si un cliente no tiene esta limitación (porque puede recibir mensajes BOOTREPLY por la unicast) este flag no debería ser activado.

El resto de los flags

El resto de los campos de los flags son reservados para usos futuros. Un cliente debería ponerlos a 0 en todos los mensajes BOOTREQUEST que genere, e ignorarlos en los mensajes de BOOTREPLY que reciba.

Definición del campo 'secs'

El campo 'secs' de un mensaje BOOTREQUEST representa el tiempo transcurrido en segundos, desde que el cliente mandó su primer mensaje BOOTREQUEST.

Uso de los campos 'ciaddr' y 'yiaddr'

Si un cliente BOOTP no conoce que dirección IP utilizará, entonces debería poner el campo 'ciaddr'=0.0.0.0 Si el cliente tiene la habilidad de recordad la última dirección IP que le fue asignada (o que haya sido preconfigurada por medio de un mecanismo alternativo), entonces debería poner en 'ciaddr' dicha dirección. Si el cliente pusiera una dirección distinta de 0.0.0.0., entonces estaría preparado para aceptar cualquier datagrama unicast direccionado a esa dirección IP. También responderá a requerimientos de ARP para esa dirección IP (si ARP es utilizado en la red).

El servidor BOOTP es libre de asignar una dirección IP diferente (en el campo 'yaddr') que el cliente expresó en 'ciaddr'. Por lo tanto el cliente debería adoptar la dirección en 'yaddr' tan pronto como fuera posible.

Interpretación del campo 'giaddr'

Este campo es raramente utilizado. Este existe para facilitar la transferencia de mensajes BOOTREQUEST desde un cliente, a través de agentes retransmisores (o rely) BOOTP, a servidores en diferentes redes que el cliente. Análogamente, éste facilita el paso de mensajes BOOTREPLY de vuelta desde los servidores (a través de agentes retransmisores de BOOTP), al cliente. Pero no hay que confundirlos con un router de IP general que es

utilizado por el cliente. Un cliente BOOTP debería poner 'giaddr' a 0.0.0.0 en todos los mensajes BOOTREQUEST que genera.

Información del vendedor "magic cookie"

Se recomienda que un cliente BOOTP siempre rellene los primeros cuatro octetos identificadores del paquete, a los cuales se les denomina "magic cookie". Un cliente BOOTP debería hacer esto incluso si este no tuviera una información especial a comunicar al servidor BOOTP usando el campo 'vend'. Esto ayuda al servido que formato de información de vendedor debería utilizar en sus mensajes BOOTREPLY.

Si ningún magic cookie de vendedor específico es usado, el cliente debería utilizar el siguiente valor decimal en notación puntuada 99.130.83.99. En este caso, si el cliente no tiene información que comunicar al servidor, el octeto inmediatamente siguiente al magic cookie debería poner la etiqueta de fin 255 y los restantes octetos del campo 'vend' tendrían que ser puestos a 0.

4.- Comportamiento del servidor

Recepción de los mensajes BOOTREQUEST

Todos los mensajes UDP recibidos cuyo puerto UDP de destino BOOTPS (67) del cliente, son considerados para el procesamiento del servidor de BOOTP.

Host y routers son normalmente requeridos para desechar "silenciosamente" muchos datragramas que contienen direcciones fuentes IP ilegales. Una de estas direcciones ilegales es la 0.0.0.0. Sin embargo, host o routers que soportan un servidor BOOTP, deberían aceptarlas para la deliberación local de los mensajes BOOTREQUEST cuya dirección es la 0.0.0.0. Los mensajes BOOTREQUEST para direcciones legales IP también deberían ser aceptadas.

Un servidor BOOTP debería desechar cualquier mensaje UDP recibido cuyo puerto UDP destino fuera BOOTPC (68).

Uso del campo 'ciaddr'

Ha habido varias interpretaciones del cliente sobre el campo 'ciaddr'. Un servidor BOOTP debería estar preparado para olvidar todas esas interpretaciones. En general, este campo sólo no debería ser fiable como clave para identificar al cliente, sino que se usan conjuntamente los campos 'ciaddr', 'chaddr', 'htype' y 'hlen' (y otras informaciones como los campos 'vend' y 'file') para conjuntamente, decidir como responder al cliente dado. Los servidores de BOOTP deberían preservar el contenido del campo 'ciaddr' en los mensajes de BOOTREPLY; incluso igualarlos en los mensajes de BOOTREQUEST.

Estrategia para el reparto de mensajes BOOTREPLY

Una vez que el servidor de BOOTP ha creado un mensaje BOOTREPLY apropiado, éste debe ser enviado al cliente, para ello el servidor primero chequeará el campo 'ciaddr'. El puerto de destino debe ser puesto en BOOTPC (68). Sin embargo, el servido debe estar atento a problemas de identificación, debido que el servidor puede elegir ignorar el campo 'ciaddr' y actuar como si este estuviera a 0.0.0.0 para después continuar con el siguiente algoritmo:

El servidor tendrá que chequear el campo 'giaddr'. Si este campo no está a cero, el servidor debería mandar el BOOTREPLY como una IP unicast a la dirección IP identificada en el campo 'giaddr'. El puerto de destino UDP tendría que ser activado al BOOTPS (67). Esta acción mandará el mensaje BOOTREPLY directamente al agente de transferencia BOOTP más cercano al cliente. Si el servidor de BOOTP tiene conocimiento de que un cliente en particular no puede recibir mensajes BOOTREPLY unicast, entonces el servidor debería de activar un nuevo flag BROADCAST para indicar que los agentes de transferencia debería mandar el mensaje BOOTREPLY por medio del canal BROADCAST. En otro caso, el servidor debería preservar el estado del flan BROADCAST así para que el agente retransmisor pueda actuar correctamente sobre este.

Si el campo 'giaddr' es puesto a 0.0.0.0, entonces el cliente se encuentra en la misma red que el servidor. El servidor debería examinar el flag de BROADCAST, si este está a uno, o el servidor sabe que el cliente no puede recibir mensajes BOOTREPLY por unicast, la respuesta debería ser mandada como una IP broadcast usando la dirección IP de broadcast limitada 255.255.255.255 como la dirección destino IP. Si el flag es puesto a 0, la respuesta debería ser mandada como un paquete unicast IP a la dirección especificada en el campo

'yiaddr' y la dirección del nivel de enlace especificada en el campo 'chaddr'. Si no es posible unicasting, la respuesta debería ser mandada por la dirección IP limitada 255.255.255.255 como la dirección IP de destino. En cualquier caso el puerto de destino debe ser BOOTPC (68).

5.- BOOTPTAB: El fichero de configuraciòn

El fichero bootptab es el fichero de configuración de la base de datos necesaria para la asignación de direcciones IP del demonio bootpd. Su formato utiliza símbolos de dos caracteres para las etiquetas que representan bs parámetros del host. Estos parámetros están separados por dos puntos (:)

Nombre_del_cliente:tg=valor . . . tg=valor

También puede existir una entrada patrón la cual está compuesta por un nombre de host inválido y que comienza por punto (.). Esta zona es la utilizada por las demás entradas por medio de la etiqueta tc=.defecto.

Las etiquetas que afectarán a nuestra instalación son:

Sm	La máscara de subred del host
Gw	Lista de direcciones de Gateway
Vm	Selección del formato de paquetes a utilizar
Ht	Tipo de hardware del host
Hd	Directorio donde se encuentra la aplicación a descargar
Bf	Fichero de arranque
Ds	Lista de los servidores de nombres
Hn	Mandar el nombre del host al cliente
Ha	Dirección hardware del host
Ip	Dirección IP del host
Tc	Puntero a otra tabla

Un ejemplo de fichero de configuración de la tarjeta en el host, es el descrito en la instalación del entorno de desarrollo de MaRTE OS.

3.3 EVALUACION DE MaRTE O.S.

3.3.1 Introducción

Aunque existen muchas implementaciones de sistemas operativos en tiempo real y kernels que comprenden al estándar POSIX (LYNX, HP-RT, VxWORKS, QNX, etc), ninguno de ellos están disponibles para la mayoría de las plataformas de propósito especial usadas en los sistemas empotrados, por ejemplo los microcontroladores. Dichas implementaciones tampoco proporcionan el código fuente, y por tanto no pueden ser utilizados como un vehículo para la búsqueda y testeo de nuevas teorias sobre la planificación de hebras y servicios de tiempo real. El kernel de tiempo real RTEMS podría haber sido un buen candidato para tal propósito porque sus fuentes estuvieran disponibles; sin embargo, aunque ofrece un interfaz POSIX, su diseño interno no fue realizado siguiendo el modelo de hebras de POSIX, con lo que esto conlleva en el sentido de la portabilidad de las aplicaciones.

Debido a todos los factores anteriores, el Grupo de Tiempo Real de la Universidad de Cantabria decidió diseñar e implementar un kernel de tiempo real para aplicaciones embebidas que pudiera ser usado en diferentes plataformas (incluyendo microcontroladores), y que siguiera el subconjunto de normas POSIX.13 PS5.0. Este kernel (ya implementado) servirá como un vehículo para el desarrollo de aplicaciones de tiempo real y como una herramienta de búsqueda sobre la cual se puedan prototipar nuevos interfaces para Sistemas Operativos, tales como planificadores de tiempo real definidos por el usuario, controladores de interrupciones, etc.

En lo que sigue se estudiará mas a fondo dicho Sistema Operativo de Tiempo Real. El análisis ha sido dividido en dos fases:

- Un estudio teórico-técnico: sobre las consideraciones de diseño y de arquitectura del Sistema Operativo.

- Una evaluación práctica: se realizan una serie de test que son analizados y de los cuales se obtienen una serie de valores.

3.3.2 Evaluación técnica de MaRTE O.S.

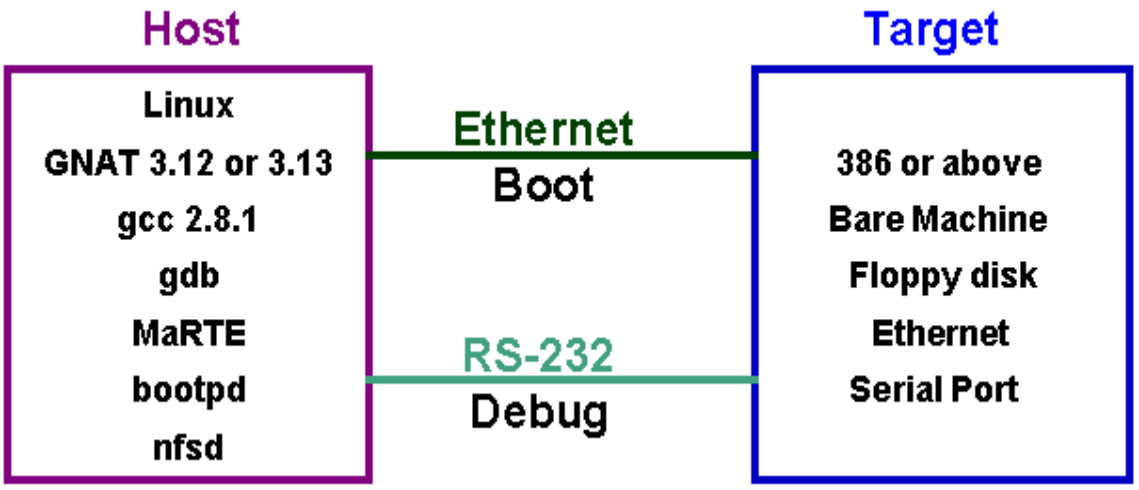
1.- Entorno de ejecución y desarrollo

MaRTE OS trabaja en un entorno de desarrollo híbrido HOST/TARGET. La computadora Host es un PC Linux con los compiladores gnat y gcc, y el depurador gdb. La plataforma de la Tarjeta es cualquier PC 386 o superior, con una disquetera necesaria para el arranque inicial del entorno de red (no es requerido disco duro).

Para la descarga remota de aplicaciones, es necesaria una tarjeta ethernet. Una línea serie puede ser utilizada para la depuración remota de las mismas.

Los protocolos bootp y nfs son usados para el arranque y descarga remota de aplicaciones desde el Hosts, a través de la tarjeta ethernet.

La aplicación final puede ser también cargada directamente desde una disquetera (o un dispositivo equivalente tal como una memoria flash), sin tener que utilizar tarjeta ethernet.



El ciclo de desarrollo y ejecución de las aplicaciones en éste entorno es el siguiente:

1.- Compilar y linkar las aplicaciones en el puesto HOST por medio de los comandos mgatmake o mgcc.

2.- Arrancar el TARGET por medio de un boot floppy (generado en el proceso de instalación de MaRTE OS).

3.- En este punto, el programa de arranque descarga desde el HOST el ejecutable de la aplicación a través de la tarjeta Ethernet, y lo lanza a ejecución.

4.- Si la aplicación hubiera sido escrita para soportar la depuración remota, entonces en este punto se podría depurar desde el HOST (de una forma sincronizada). Si no es así, entonces la aplicación se ejecutará libremente.

5.- En el instante en que la aplicación finalice su ejecución, el programa de arranque toma de nuevo el control de la computadora del TARGET y se vuelve al punto 1, pero sin tener que pasar por 2.

Objetivos que persigue

MaRTE OS está siendo desarrollado para conseguir la siguiente estandarización Posix: Cumple completamente con el perfil POSIX.13-PSE5.0, e implementa algunos servicios de POSIX.1d y POSIX.1j, los cuales son muy importantes para las aplicaciones de tiempo real que pueden ejecutarse en MaRTE OS. Estos servicios son: relojes y timers en tiempo de ejecución y un sleep absoluto de alta resolución.

Además, MaRTE O.S. ha sido diseñado para cumplir con los siguientes requerimientos funcionales:

- Tarjeteado para aplicaciones con el número de hebras y recursos de sistema bien conocidos en tiempo de compilación. Esto proporciona a esos recursos (por ejemplo hebras, mutexes, pilas de la hebras, numero de niveles de prioridad, timers, etc) ser situadas en memoria en tiempo de configuración, así se ahorra mucho tiempo cuando una aplicación requiere la creación de uno de esos objetos.

- Todos los servicios tienen tiempo de respuesta limitados, para la realización de sistemas de tiempo real duros.

- No hay protección. Por propósito de eficiencia, no habrá límites de protección establecidos entre las aplicaciones y el kernel. Esto significa que una aplicación pueda corromper a los datos del kernel, pero esto no debería ser un problema en sistemas testeados estáticamente, como las aplicaciones tarjeteadas

- Multiplataforma. El kernel será implementado para múltiples plataformas de tarjetas, usando herramientas a lo largo del desarrollo.

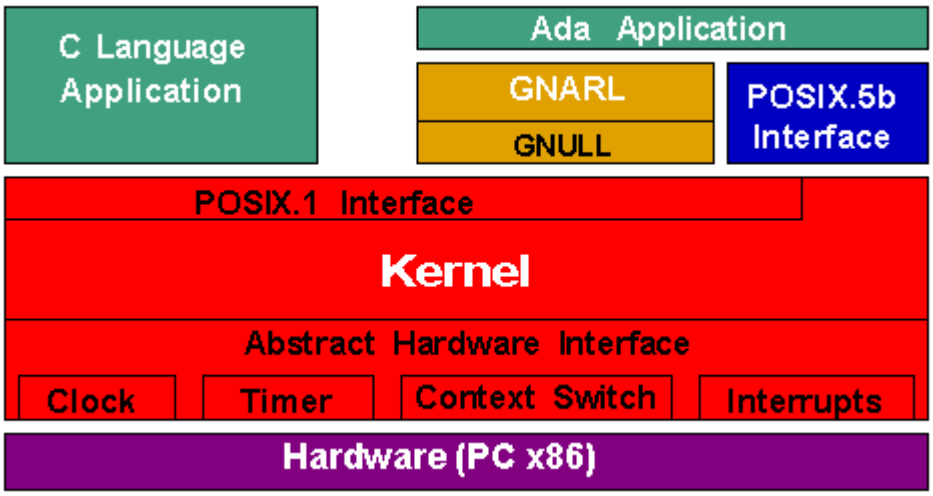
- Multilenguaje: el kernel soportará aplicaciones escritas en Ada y C e incluso aplicaciones con ambos. En el futuro, se añadirá la máquina virtual de Java, para también soportar aplicaciones en tiempo real escritas en Java.

2.- Arquitectura del kernel de MaRTE O.S.

La arquitectura del kernel soporta aplicaciones en Ada y C para el desarrollo usando los compiladores GNU gnat y gcc. Casi todo su código está escrito en Ada con algunas partes en C y ensamblador. Aplicaciones en C con multihebras pueden ser realizadas usando el kernel escrito en Ada, debido a que se ha implementado un interfaz para este lenguaje.

El kernel posee un interfaz abstracto de bajo nivel para acceder al hardware. Este interfaz encapsula operaciones para el manejo de interrupciones, manejo del clock y timers y cambios de contexto de las tareas. Su objetivo es facilitar la migración desde una plataforma a otra. Para hacer dicha migración, se modificarán dichos interfaces por una implementación adaptada al hardware deseado.

Para plataformas basadas en PC algunas de las funciones de este interfaz abstracto del hardware vienen desde un conjunto de herramientas públicas disponibles llamadas OSkit, el cual está intentando facilitar los aspectos de bajo nivel del desarrollo de un sistema operativo. Estas están escritas en los lenguajes C y ensamblador. También utilizamos las utilidades de OSKit para el arranque de aplicaciones desde un disquete o desde la red.



Interfaz para las aplicaciones Ada en MaRTE OS

El kernel es directamente utilizable como base para el entorno de ejecución de Gnat. Gracias a ello, se podrán programar aplicaciones Ada usando las tareas específicas del lenguaje.

Dicha ejecución es posible gracias a que el compilador de gnat es un software libre y proporciona las fuentes, incluyendo su núcleo de ejecución del sistema llamado GNARL; esto es extremadamente importante ya que se necesitan modificar partes de este núcleo de ejecución para adaptarlo al kernel de MaRTE OS.

3.- La A.P.I. de P.O.S.I.X.

Actualmente MaRTE OS implementa las siguientes funcionalidades POSIX. Ellas cubren el perfil mínimo de POSIX.13 más algunas extensiones desde POSIX.1j y POSIX.1d:

- Manejador de Hebras de POSIX: Creación de hebras, finalización, atributos, etc. Todas las hebras y pilas de las hebras son preasignadas, así la creación de las hebras es extremadamente rápida. El número de las hebras y de los tamaños de las pilas son configurables.

- Planificador con Prioridades: Se han implementado las dos políticas de planificación exclusivas requeridas por POSIX: FIFO con prioridades, y Round Robin con prioridades.

- Mutexes: Utilizados para la exclusión mutua entre hebras. Se han implementado la herencia de prioridad y techos de prioridad inmediata, para evitar las inversiones de prioridades ilimitadas.

- Variables de Condición: Para la sincronización de esperas condicionales.

- Señales: Estas son el mecanismo básico untilizado para la notificación de eventos. En particular, son usadas por los timers para notificar sobre sus expiraciones de tiempo.

- Clocks: Se implementa el clock de tiempo real de POSIX para la medida del tiempo.

- Timers: Son objetos software que miden intervalos de tiempo o detectan cuando un reloj alcanza un tiempo acordado.

- Clocks y Timers de Tiempo de Ejecución: Llevan la cuenta del consumo y cantidad de tiempo de CPU. Es una herramienta extremadamente interesante para detectar y limitar las sobrecargas de la CPU en aplicaciones de tiempo real, que podrían invalidar el resultado del análisis de la planificación de un conjunto de tareas dado.

- Consola de E/S: Para la visualización de E/S usando la consola. Otros drivers de E/S serán proporcionados en el futuro.

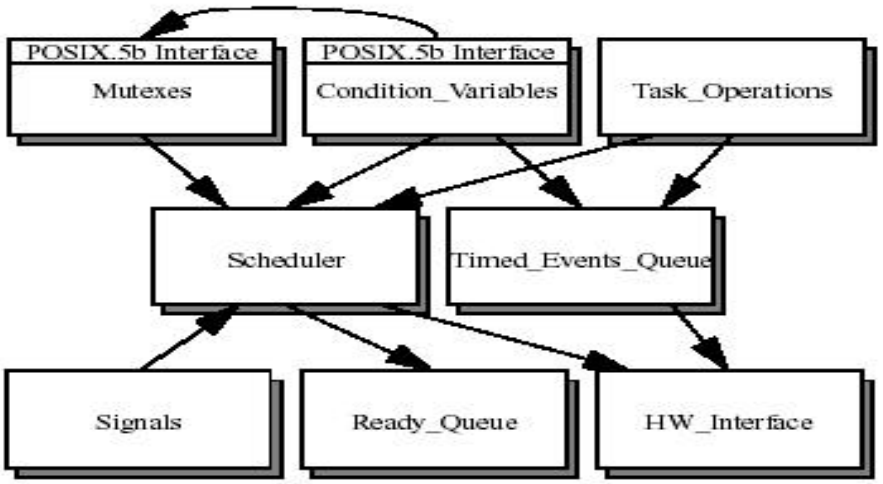
- Servicios Temporales: Se han implementado la suspensión de hebras con delays absolutos y relativos de alta resolución. Los delays absolutos todavía no son parte de POSIX.13, pero se han incluido debido a que son extremadamente útiles; y si no un timer muy costoso tendría que ser utilizado.

- Manejo de Memoria Dinámica: Aunque no es parte del interfaz de POSIX, esta funcionalidad ha sido implementada debido por los requerimientos de los lenguajes de programación.

Todas estas funcionalidades POSIX implementadas son necesarias para suportar todas las librerías del núcleo de ejecución de Gnat. Por lo tanto la implementación de MaRTE OS facilita la ejecución de aplicaciones complejas escritas en Ada y C, con la restricción de que

ellas no usen sistemas de ficheros. Esta limitación es impuesta por el perfil del subconjunto de tiempo real mínimo POSIX.13, el cual es el que sigue MaRTE OS.

En la siguiente figura, se puede apreciar la interconexión entre las funcionalidades implementadas en Posix.



4.- Herramientas disposibles

MaRTE OS trae consigo un conjunto de herramientas para la programación y testeo de aplicaciones de usuario:

- Depuración remota: Depurador de programas por medio de la red.

- Inspector de tareas: El cual sirve para recoger información de los procesos que se han ejecutado en el sistema.

- API para crear algoritmos de planificación: Con esta herramienta se pretende poder implementar algoritmos de planificación de una forma predecible y compatible con el estándar de Posix.

- API para el manejo de interrupciones: Facilita la programación de interrupciones externas.

- Manejadores de consolas: Básico pero suficiente conjunto de librerías para la visualización por pantalla.

Este conjunto de herramientas es suficiente para poder trabajar con el entorno de desarrollo de MaRTE O.S. de una forma sencilla.

CAPITULO 4. Evaluación práctica de MaRTEO.S.

4.1 PLANIFICACION DE TAREAS

Los programas concurrentes, a diferencia de los programas secuenciales, se caracterizan por tener varios procesos que realizan su trabajo de forma concurrente o alternada. Si un programa es correcto, entonces la salida obtenida al final de la ejecución siempre será la misma, aunque las trazas de ejecución no tienen por que serlo, en otras palabras:

sea un conjunto de n tareas, si se lanzan a ejecutar en un sistema mono-procesador, existen X formas diferentes de ejecución que obtienen el mismo resultado. Si se ejecutan en un sistema multi-procesador, existirán muchas más, donde:

$$X = (N * M)! \ / \ (M !)^{n}$$

Cuando trabajamos con sistemas en tiempo real, el orden en que se ejecutan las tareas es muy importante, debido a la necesidad de cumplir los límites de tiempo, o deadlines, impuestos a cada proceso. Es decir, de las X trazas de ejecución posibles, puede que solo unas cuantas cumplan dichas restricciones de tiempo. A este estudio se le denomina: Planificación de Procesos.

La planificación de procesos consta de dos partes:

- Un algoritmo para la ordenación de los recursos del sistema, en particular la CPU.
- Un estudio para predecir el peor caso de ejecución del conjunto de tareas que se han planificado.

Una vez planificadas las tareas, estaremos en condiciones de ver si los requerimientos temporales del sistema han sido cumplidos o no, y por lo tanto el programa será valido o no.

Pero para realizar todo este estudio, antes de todo habrá que conocer muchos parámetros internos del entorno de ejecución donde se va a ejecutar. Este es el objetivo final del proyecto:

"La obtención de los parámetros temporales necesarios para la planificación de un conjunto de tareas sobre MaRTE.OS".

Se podrían estudiar muchos parámetros temporales dependiendo de la naturaleza de los procesos, recursos del sistema a utilizar, el entorno de ejecución, etc, pero este estudio no es trivial, debido a que normalmente los parámetros están estrechamente relacionados con el hardware donde se van a ejecutar los procesos. Normalmente se suele hacer un estudio particular para cada tipo de hardware a utilizar, por lo tanto, el objetivo final del proyecto será obtener los parámetros temporales necesarios para la siguiente configuración:

- Hardware del Target: Intel Pentium 120

- Sistema Operativo del Target: MaRTE OS

- Tipo de Tareas de las aplicaciones: Tareas periódicas, con prioridades estáticas, planificación expulsiva, con comunicación y sin ella.

1.- Notación utilizada

La notación estándar que será utilizada para el análisis temporal de las tareas es:

B	Peor caso del tiempo para el proceso bloqueado
C	Peor caso del tiempo de computación del proceso
D	Deadline de los procesos
I	El tiempo de interferencia para los procesos
J	Jitter de activación de un proceso
N	Número de procesos en el sistema
P	Prioridad asignada a un proceso
R	Tiempo de respuesta del proceso
T	Mínimo tiempo entre activaciones de un proceso (periodo)
U	Tiempo de utilización de cada proceso (igual a C/T)

2.- Planificación basada en procesos

Para poder planificar los procesos será necesario modelarlos de alguna forma. Este estudio estará basado en el modelo de ejecución utilizada por los sistemas operativos generales, el cual define los siguientes estados de un proceso:

- Listos para ser ejecutados
- En ejecución
- Suspendido esperando un evento temporal
- Suspendido esperando un evento no temporal

Los procesos listos para ser ejecutados son ejecutados de acuerdo con sus prioridades, las cuales suelen asignarse por sus requisitos temporales, no por su importancia o integridad. Cuando un proceso de menor prioridad se está ejecutando y existe otro de mayor prioridad que está listo para ser ejecutado, pueden ocurrir dos cosas: que se cambie directamente a ejecución el proceso de mayor prioridad, o que termine el proceso que se está ejecutando y luego se ejecute el de mayor prioridad. Dependiendo de este factor, distinguimos entre planificación expulsiva y no expulsiva.

3.- Asignación de prioridades monótonas

Una forma de indicar en el orden que se van a ejecutar los procesos, es por medio de la asignación de prioridades. Las prioridades determinarán que proceso va a ser ejecutado en cada punto de planificación. Nosotros vamos a utilizar el algoritmo de asignación de prioridades monótonas.

Éste es un algoritmo óptimo, es decir, dado un conjunto de procesos, si existe una traza que cumple todas las restricciones de tiempo entonces es posible obtenerla aplicando dicho algoritmo. Aquí las prioridades son asignadas de acuerdo con los periodos de los procesos siguiendo la siguiente propiedad:

$$T_i < T_j \Rightarrow P_i > P_j$$

Una vez hecho esto, nos interesa ver si dicha asignación de prioridades cumple las restricciones de tiempo (sus límites de tiempo). Para ello, se realizará el siguiente estudio:

1.- Un estudio analítico para predecir el peor tiempo de respuesta para cada proceso.

2.- Se comparan dichos tiempos con los deadlines de las tareas para ver si cumplen los requisitos temporales del sistema. Si se cumplen todos los deadlines entonces el sistema es planificable para dicho conjunto de tareas. Por el contrario, si alguno no cumple el deadline asociado, entonces el sistema no será planificable.

Para ello se utilizará la siguiente fórmula:

$$R_i = C_i + I_i$$

donde con I se representa la interferencia en el tiempo de ejecución de la tarea i debido a la ejecución de otras tareas de mayor prioridad. Y con C su peor tiempo de ejecución cuando se ejecuta sola.

Nota.- Esta expresión está muy simplificada, ya que no tiene en cuenta las comunicaciones entre procesos, los cambios de contexto y otros parámetros muy importantes a la hora del análisis temporal. En la siguiente sección se definirá y comentará la fórmula que se va a utilizar en nuestro análisis.

4.- Interacción y bloqueo entre procesos

Un aspecto muy importante a la hora de planificar procesos es la comunicación entre éstos (por medio de variables protegidas compartidas, semáforos, objetos protegidos o simples citas). Todas estas herramientas de los lenguajes de programación, suelen suspender a los procesos involucrados hasta que se cumplan ciertas condiciones, como por ejemplo, dos procesos que quieren comunicarse por medio de una cita, se tienen que encontrar en el lugar de su código donde se realiza dicha cita.

Pero cuando se utilizan prioridades, puede ocurrir un problema muy grave: la inversión de prioridad. Este problema puede llevar a una tarea de alta prioridad a ser bloqueada por tiempo no limitado por otra de menor prioridad. Esto es inadmisible en un S.T.R., pero si ocurriera, tendríamos que poder medirlo y sobre todo limitarlo.

Para limitar el efecto se utiliza el algoritmo de herencia de prioridad. En este algoritmo, la prioridad de los procesos se vuelve dinámica, y va cambiando durante la ejecución del proceso cada vez que éste accede a un recurso que puede ser tomado por otro proceso de mayor prioridad.

Otro problema muy importante es el interbloqueo, el cual será evitado con la aplicación de los siguientes algoritmos.

Un dato más a tener en cuenta, a la hora de aplicar estos algoritmos, es la sobrecarga del planificador. Debido al trabajo extra que tendrá que realizar para la reasignación de las prioridades.

Una posible aproximación, para poder medir los tiempos de respuesta para las tareas teniendo en cuenta estas circunstancias será:

$$R_i = C_i + B_i + I_i$$

También es posible modelar el límite máximo de bloqueo de un proceso por otro de menor prioridad. Para obtener este tiempo, se puede hacer por medio de la siguiente fórmula:

$$B_i = \sum_{K=1}^{k} usage(k,i)\ CS(k)$$

Donde:

- usage(k,i) es una función {0,1} que indica que el recurso utilizado por la tarea i, también será utilizado por otras tareas.
- CS(K) es el tiempo de cómputo del recurso.

Lo que ocurre es que esta aproximación es bastante pesimista, debido a que una tarea solo será bloqueada en un recurso si y solo si lo utiliza. Pero es muy útil para poder definir algoritmos nuevos y hacer una estimación a priori del análisis temporal del conjunto de tareas, ya que cumple que es una condición necesaria pero no suficiente.

Protocolos de techos de prioridad

Surgen con la necesidad de aproximar los valores de Bi de una forma más exacta. Además, la aplicación de estos algoritmos evitan el interbloqueo por definición.

Existen dos algoritmos definidos: techos de prioridad original y el inmediato. Ambos se comportan igual con respecto al parámetro Bi, lo que ocurre es que uno realiza mas cambios de contexto que el otro.

El cálculo del tiempo de bloqueo de una tarea por otra de menor prioridad, cuando ambas acceden al mismo recurso es:

$$B_i = \max_{\forall k \text{ å } lp(i),\ \forall r \text{ å } usa(k) \mid techo(r) >= pri(i)} CS_{k,r}$$

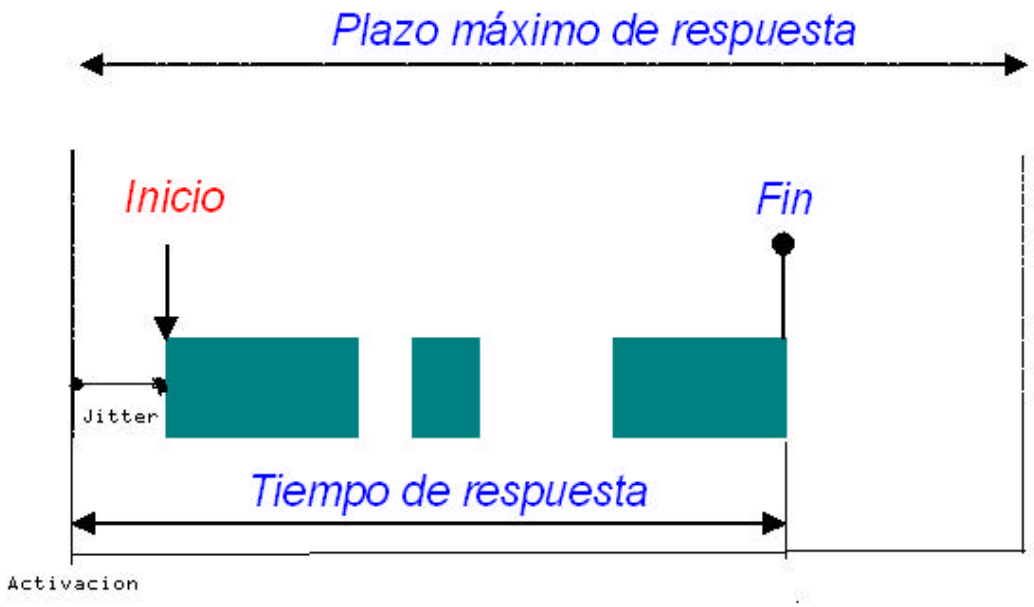
Donde se seleccionan los recursos accedidos por tareas de menor prioridad que i, y de éstos solo aquellos que son accedidos también por la tarea i y que tengan mayor techo de prioridad. De los recursos seleccionados, se da como resultado el tiempo de ejecución mayor entre todas las secciones críticas de cada recurso.

5.- Otros parámetros de planificación

Jitter de activación

Otro problema aparece cuando trabajamos con tareas periódicas es el Jitter de activación. Sería un error pensar que toda tarea periódica comienza a ejecutarse en el mismo instante que es planificada, debido a que existen muchas operaciones internas al procesador que deben ser ejecutadas.

Por lo tanto, sería necesario tenerlo en cuenta, debido a que si a cada tarea se le añade un jitter, puede ocurrir que a lo largo del tiempo este lleve al conjunto de tareas a no ser planificable.



Cambios de contexto

Este es un parámetro interno es clave para cualquier microprocesador, y como su propio nombre indica, es la operación el sistema realiza para llevar una tarea de la cola de preparados a ejecución, y llevar a la tarea que actualmente se esté ejecutando a la cola de bloqueadas o preparadas, según sea el caso.

Por lo tanto entre los trabajos a realizar se encuentran:

- Guardar los registros de estado de la tarea actualmente ejecutándose.

- Cargar los registros de estado de la tarea que sea seleccionada por el dispacher para ser ejecutada.

- Cambiar el contador de programa a código de la tarea a ejecutar.

El planificador

El planificador es la tarea del sistema que se encarga del control de ejecución de los procesos, asignación de recursos, etc. Por lo tanto también habrá que tenerlo en cuenta y modelarlo a la hora de hacer el estudio temporal de un sistema. Actualmente se estudian dos tipos de planificadores:

a) Planificación por eventos

En este esquema el planificador tiene dos colas de procesos:

- Cola de ejecución
- Cola de tiempos

Su funcionamiento es el siguiente:

Cuando el planificador suspende a una tarea por medio de una primitiva (por ejemplo delay until en Ada), la tarea pasa del estado de ejecución a la cola de tiempos y se programa

una interrupción de un time con el menor tiempo de activación de las tareas que se encuentran en dicha cola. Una vez que dicha interrupción ocurre, todas las tareas en la cola de tiempos con tiempos menores o iguales a dicho instante serán cambiadas a la cola de ejecución. Y se vuelve a programar dicha interrupción. En este instante el planificador ejecuta la tarea de mayor prioridad de la cola de ejecución.

Como se puede apreciar, el planificador introduce una pequeña sobrecarga al sistema. Se puede modelar dicha sobrecarga como una tarea ficticia para cada tarea con las siguientes características temporales:

- Tiene la prioridad más alta del sistema.
- Su periodo será el mismo que el de la tarea que modela.
- Su C será igual al peor tiempo de ejecución del manejador de interrupciones del timer.

Por lo tanto podríamos tener una fórmula para los tiempos de respuesta como sigue:

$$R_i = C_i + 2\,C_{SW} + B_i + \sum_{j \in Hp(i)}^{T} \left[(R_i + J_j) / T_j \right] * (C_j + 2\,C_{SW}) + \sum_{j=1}^{T} \left[(R_i + J_j) / T_j \right] * C_{Timer}$$

b) Planificación dirigida por ticks

Es una simplificación del protocolo anterior, para evitar que el planificador tenga prioridades dinámicas, pero es más ineficiente ya que la interrupción del timer, (la cual lanza a ejecución el planificador), será generada periódicamente con un valor denominado TICK.

Por lo tanto, cada vez que salta el planificador, comprueba si existe alguna tarea con el tiempo expirado en la cola de tiempos.

Para modelar este planificador, se tomarán los siguientes criterios:

- Se crea un conjunto de tareas ficticias que representen el coste asociado a la manipulación de llevar todas las tareas de una cola a otra.

- Crear una tarea periódica con periodo igual al TICK que será el tiempo de ejecución del planificador

- Añadirle el Jitter de activación

Por lo tanto, teniendo estas consideraciones, podremos utilizar la siguiente fórmula para obtener los tiempos de respuesta para un conjunto de tareas que se ejecutan conjuntamente con el planificador descrito:

$$R_i = C_i + 2C_{SW} + B_i + \sum_{j \in Hp(i)}^{T} [(R_i + J_j) / T_j] * (C_j + 2C_{SW}) + \sum_{j=1}^{T} [(R_i + J_j + Tick) / T_j] * C_{Cola} + [R_i / Tick] * C_{tick}$$

Una vez comprendida la planificación, podremos comenzar en el diseño e implementación de los tests.

4.2 SELECCIÓN DEL CONJUNTO DE APLICACIONES PARA LA EVALUACION DE MARTE O.S.

Como se viene indicando a lo largo de todo el proyecto, la idea es hacer un estudio de los parámetros temporales necesarios para realizar un análisis de planificación por tiempos de respuesta, sobre un conjunto de tareas que se ejecutan en el microkernel de tiempo real MaRTE OS.

Los tests a realizar serán los siguientes:

G) General:

G.0) Búsqueda de la herramienta para medir tiempos del sistema.
G.1) Constantes temporales del sistema.
G.1) Comportamiento de los delays del sistema.

T) Parámetros temporales de planificaciòn

T.1) Obtención de los cambios de contexto del sistema
T.2) Jitter de activación de las tareas
T.3) Tipo de planificador que utiliza
T.4) Tiempo de despertado de las tareas

P) Tests de planificabilidad de tareas:

P.1) Selección de un conjunto de tareas independientes para estudiar su planificabilidad.
P.2) Selección de un conjunto de tareas que se comunican para estudiar su planificabilidad.

4.3 DISEÑO E IMPLEMENTACIÓN DE LOS TEST Y OBTENCIÓN DE DATOS TEMPORALES

En este apartado se van a diseñar y explicar los tests que se han realizado sobre MaRTE OS, y se comentarán los resultados obtenidos.

G) General

G.0) Búsqueda de la herramienta para medir tiempos del sistema

TEST

Antes de todo, se tendrán que estudiar las herramientas a utilizar para la toma de tiempos. Es decir, leer el reloj del sistema para ver el tiempo que transcurre al ejecutarse una acción.

Para ello siempre se hará lo mismo, tomamos la hora justo antes de cada acción, y la volvemos a leer justo después. Por lo tanto la resta de estos dos valores será la duración en tiempo de la ejecución de la acción analizada.

El problema aparece cuando la herramienta escogida para ello no tiene la precisión suficiente, por lo tanto no saldrán los tiempos tan ajustados como se pretende, pudiéndonos llevar a errores en nuestras mediciones.

Analizando MaRTE OS se puede apreciar que las formas de leer los relojes son las siguientes:

Inicialmente podría usarse:

ada.real_time.clock

Pero esta función pertenece a Ada y se ejecuta sobre el entorno de ejecución GNULL, el cual a su vez está justo encima del código del kernel de MaRTE, que a su vez llama a la rutina

Get_time

perteneciente al paquete kernel.timers, que llama a su vez a la función del interfaz con el hardware:

/x86/rtc/rtc.Get_rtc

entonces se obtienen resoluciones dejan mucho que desear, a la hora de hacer mediciones en los test.

Otra forma de tomar tiempos sería por medio de:

/gnat-3.13/system.task_primitives.operations.clock

la cual devuelve un Duration.

Aunque con esta función se obtienen resultados temporales mas precisos que con ada.real_time.clock, tampoco es muy satisfactoria.

Otras formas podrían ser:

Por medio de funciones del interfaz Posix

/kernel/time.get_time(clock_id: clock_id_t; tp: access timespect) return integer

donde:

```
type timespect is record
   Tv.sec:int; tv_nsec:int;

subtype clock_id_t is timers.clock_id;

CLOCK_REALTIME:constant Clockid_T:=
                                    timers.CLOCK_REALTIME;
```

Utilizando el temporizador de tiempo real del sistema:

/kernel/kernel.timers.get_time(clock:clock_id) return duration;

donde:

clock_id is new basic_integer_32;

CLOCK_REALTIME : CONSTANT clock_id:=
General_constan.clock_REALTIME;

Leyendo la hora directamente con la función que devuelve la hora absoluta del sistema:

/x86/hardware_interface.get_hw_time return hwtime;

donde:

subtype hwtime is TSC.procesor_cycles;

pero en realidad hace la cuenta desde el comienzo de la época, por lo tanto es bastante preciso ya que tiene que realizar la siguiente conversión:

```
function Get_HWtime return HWtime is
  begin
    return Total_Time + TSC.Read_TSC;
end Get_HWtime;
```

Gracias a que MaRTE OS es un SW abierto que proporciona los fuentes, se puede observar sus implementaciones. Por lo tanto, se puede deducir (a partir de la función anterior) que es mas eficiente utilizar directamente:

/x86/TSC/function read_tsc return Processor_cycles;

la cual lee el timer interno del micro_procesador (T.S.C), que es utilizado para implementar la hora del sistema.

La lectura del valor del TSC se hace por medio de instrucciones en ensamblador, por lo tanto es lo mas instantáneo a la hora de medir acciones, que es lo que nos interesa para nuestro trabajo.

CODIGO

En resumen, tomando el TSC como base, se puede apreciar que es una buena herramienta para la toma tiempos. Esto se puede ver al realizar siguientes medidas:

```
T1:=Tsc.read_tsc;
Función que consulta el tiempo del sistema
T2:=tsc.read_tsc;
Dur:=T2-t1;
```

TIEMPOS OBTENIDOS

Ada.real_time.clock	=	4 µs
tsc.read_tsc	=	200 ns

CONCLUSION

Todos los tests serán medidos por medio del TSC debido a que éste contador (como es indicado en la documentación de MaRTE , y la perteneciente a Intel P5), es un contador interno de 64 bits el cual está directamente "alimentado" con la señal de entrada a la CPU clk. Este contador es calibrado a la hora del arranque del sistema para adecuarse a la velocidad del procesador, por lo tanto se incrementa cada ciclo de reloj. Además es posible leer directamente sus valores de una forma atómica, es decir, en un ciclo de reloj.

G.1) Constantes temporales del sistema

TEST

Como la programación de aplicaciones en este proyecto se realizan por medio del interfaz con Ada95, entonces habrá que definir la documentación de las constantes y variables definidas para el nivel inferior del hardware sobre el que trabaja Ada95. Es decir, las que especifican el entorno de ejecución donde se ejecutan.

TIEMPOS OBTENIDOS

Los datos obtenidos para el reloj monótono son:

Sea T: time;

T_Unit = 1 nanosegundo

T_Span_Unit = 1 nanosegundo

system.tick = 1 nanosegundo

ada.real_time.tick = 1 microsegundo

CONCLUSION

Esta información no es utilizada en los tests, debido a que nosotros medimos tiempos utilizando el TSC interno del procesador. Pero es muy útil a la hora de programar aplicaciones bajo Ada95.

G.2) Delays

Para poder trabajar con las tareas periódicas, hay que utilizar una instrucción que permita dormir a la tarea hasta su próxima activación. Nuestros tests irán enfocados al delay absoluto (delay until), porque el delay relativo (delay) no es muy útil a la hora de programar aplicaciones de tiempo real, debido a su granularidad y otros aspectos internos de implementación.

Las medidas de los delays a documentar son:

G.2.0) Tiempo de ejecución de la expresión delay until cuando el bloqueo no es realizado.

TEST

Consistirá en la ejecución repetida de una única tarea que contiene la instrucción delay until, con un valor nulo.

CODIGO

```
Periodo:= 0.0;
Next_time:= ada.real_time.clock;

T1:=tsc.read_tsc;

Delay until next_time;
Next_time:= next_time + periodo;

T2:=tsc.read_tsc;

Dur:=t2 – t1;
```

TIEMPOS OBTENIDOS

Los tiempos obtenidos están comprendidos en el intervalo:

(39’421 , 40’811) µseg.

CONCLUSION

La cota superior del tiempo consumido por la operación delay until, cuando la hora de activación ya ha expirado es:

41 µseg.

Este dato es muy importante a tener en cuenta en la realización de nuestros test, ya que están compuestos principalmente por tareas periódicas, este dato siempre estará presente.

G.2.1) Diferencia mínima entre el valor del clock y la expresión del delay until.

TEST

Consistirá en ejecutar una sola tarea e ir testeando el valor del periodo, hasta conseguir uno que siempre lleve a la tarea a suspenderse.

CODIGO

```
Periodo:= X.X;
Next_time:= ada.real_time.clock;

T1:=tsc.read_tsc;

Delay until next_time;
Next_time:= next_time + periodo;

T2:=tsc.read_tsc;

Dur:=t2 – t1;
```

TIEMPOS OBTENIDOS

El valor correcto será cuando siempre se duerma la tarea, es decir que su tiempo mayor o igual que Periodo.

Después de haber realizado algunas pruebas para valores del periodo comprendidos entre:

(40 , 200) µseg

y se ha obtenido que el valor buscado es:

15O µseg.

También se ha realizado el test con 26 tareas que sobrecargan al sistema y se ha obtenido que el valor es:

155 µseg.

CONCLUSION

El valor mínimo permitido como periodo para una tarea periódica, y que el sistema funcione correctamente es:

155 µseg

Hay que tener en cuenta, que este valor es posible utilizarlo solo cuando las tareas carecen de código (ya que al introducir código forzosamente la tarea sobrepasará dicho valor).

Por lo tanto, se ha obtenido la cota mínima de periodo del sistema para las tareas periódicas, la cual nunca se llegará a utilizar.

T) Parámetros Temporales de planificación

Esta serie de tests están enfocados a la obtención de los parámetros temporales de planificación (más relevantes) pertenecientes a MaRTE OS.

T.1) Cambios de contexto

Los cambios de contexto son unas operaciones que siempre están presentes en el sistema, por lo tanto obligatoriamente habrá que estudiarlos.

TEST

Existen varias formas de estudiarlos, aunque la más intuitiva consiste en: crear una tarea de baja prioridad que lea continuamente el reloj del sistema y añadir una tarea de alta prioridad (sin cuerpo) que interrumpa periódicamente.

Con esta situación, será inmediata la obtención del valor asociado al cambio de contexto. Este valor podrá ser definido como:

"el tiempo que tarda en leer el reloj la tarea de menor prioridad cuando es interrumpida por la de mayor prioridad y dividido entre dos".

Dividir entre dos es debido a que una tarea realiza dos cambios de contexto cuando interrumpe a otra. Uno es cuando interrumpe y el otro es cuando abandona al procesador, y así proseguir la ejecución de la tarea interrumpida.

CODIGO

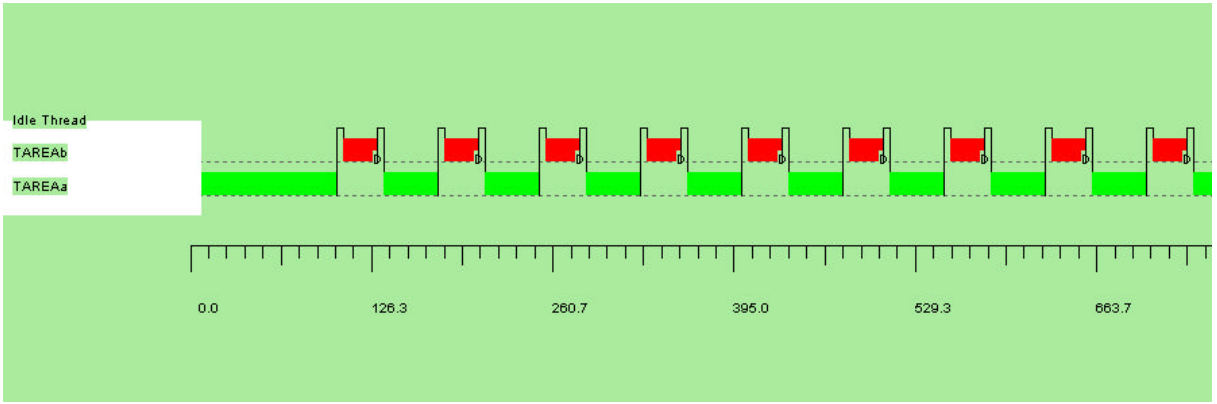
Tarea lectora del reloj del sistema

```
for i in 1 .. N loop
        t1:= get_hwtime;
        t2:= get_hwtime;
        almacenar_tiempos( arr_tiempos, t1, t2);
end loop;

calcular_tiempos( arr_tiempos);

visualizar_intervalos( arr_tiempos);
```

Tarea que expulsa a la lectora

```
next_time := clock + periodo;
for i in 1 .. M loop
        delay until next_time;
        next_time:= next_time + periodo;
end loop;
```

CRONOGRAMA



TIEMPOS OBTENIDOS

La traza temporal obtenida es:

(. . . 199, 199, 366, 224, 199, 199, 127_266, 224, 199, 199 . . .) nanosegundos

Sobrecargando al sistema con 26 tareas adicionales de mayor prioridad, las cuales se bloquean, se obtiene la siguiente traza:

(. . . 199, 199, 366, 224, 199, 199, 140_360, 224, 199, 199 . . .) nanosegundos

CONCLUSION

Por lo tanto, el tiempo de cambio de contexto cuando el sistema está sobrecargado (nos interesa mas por darnos una cota superior más segura) es:

140 µs / 2 = 70 µs

Pero al valor original obtenido, habría que restarle el tiempo de ejecución de la instrucción delay until (40 µs), por lo tanto el tiempo de cambio de contexto sería:

(140 – 40) µs / 2 = 50 µs

T.2) Jitter de activación

TEST

El Jitter de activación, es el tiempo que tarda el dispacher del sistema operativo en pasar una tarea que está en la cola de preparados a ejecución.

En este test, se ha optado por meter a todas las tareas en la cola de ready, para que vayan pasando a ejecución ordenadamente. Cada tarea se bloqueará inmediatamente una tras otra. Esto permitirá obtener los tiempos que transcurren desde el instante en que una tarea deja el procesador y la tarea siguiente lo toma.

CODIGO

Se creará un conjunto de tareas con la misma prioridad, y con el cuerpo siguiente:

```
Periodo := X;
Next_time:= ada.real_time.clock;

For i in 1 .. N loop

        Delay until next_time;

        T:=tsc.read_tsc;

        Next_time:= next_time + periodo;
End loop;

Dur := T_tarea_actual – T_tarea_anterior
```

TIEMPOS OBTENIDOS

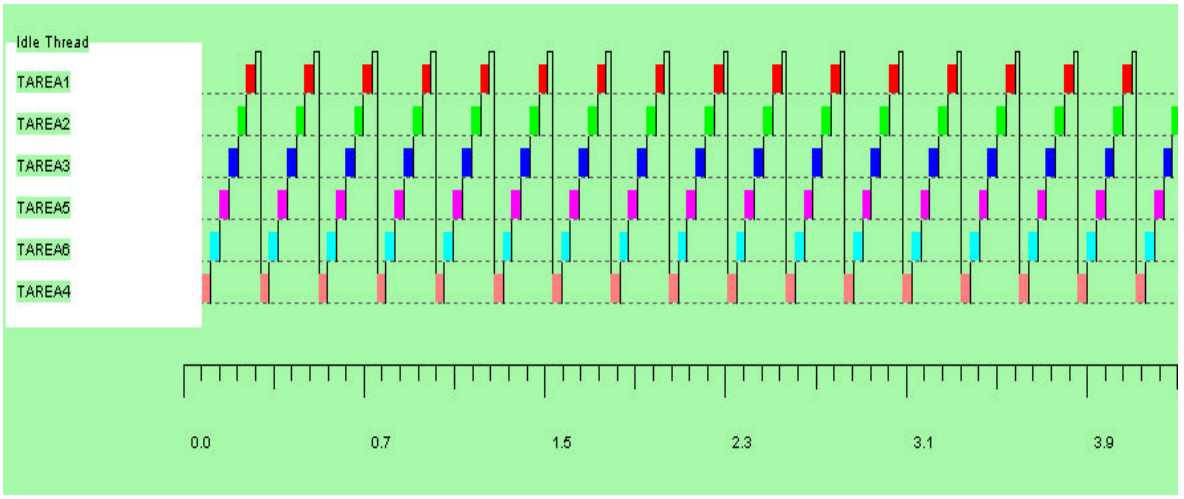
Los datos obtenidos al hacer restas (entre el instante en que una tarea deja al procesador y otra llega a conseguirlo), se encuentran entre los siguientes valores:

(50’895, 75’527) µsegundos

aunque la mayoría se encuentran entre:

(50’895, 52’044) µsegundos

CRONOGRAMA



CONCLUSION

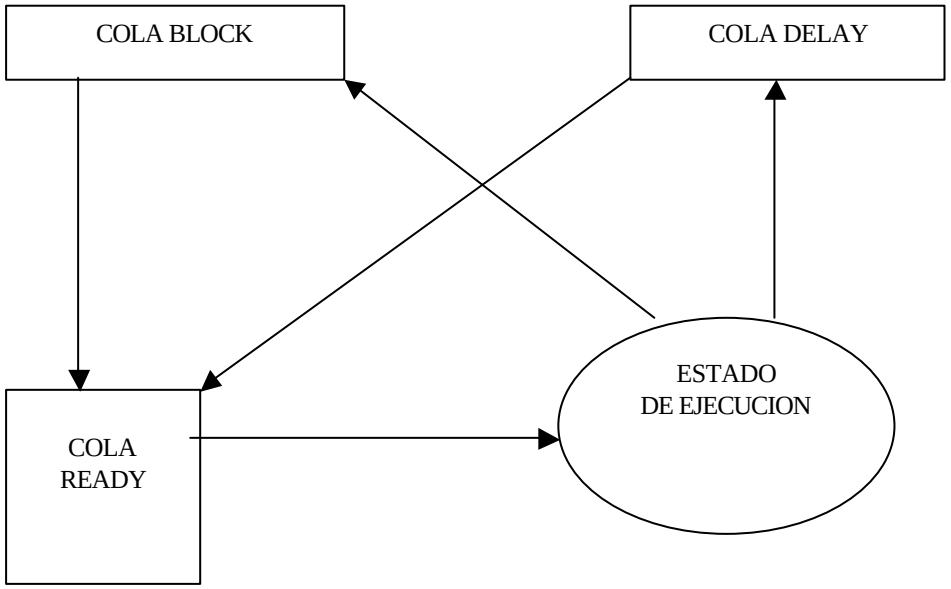
A los datos anteriores, habría que restarle el tiempo de ejecución de la instrucción delay until, que como se analizó en G.1, es del orden de 40 µseg.

Por lo tanto, se puede deducir que el tiempo que tarda el dispacher en llevar una tarea encolada en Ready a Ejecución es:

52 – 40 = 12 µseg

A veces a parecen valores del orden de 75 µsegundos. Esto es debidos a operaciones internas del procesador que por ahora no pueden ser definidas.

De este test también se obtiene, que la forma de levantarse las tareas cuando son dormidas hasta una hora común, sigue una ordenación HEAP. Esto nos puede llevar a la conclusión de, que la cola utilizada por los delays no es la misma que la cola utilizada cuando una tarea de alta prioridad bloquea a otra. Una forma gráfica de representarlo seria:



T.3) Tipo de planificador que utiliza MaRTE OS

En este punto nos encontramos analizando el dato más importante de un sistema operativo: el planificador. A nosotros nos interesa conocer si el planificador que utiliza sigue el modelo de planificación dirigida por ticks o por eventos.

TEST

Como se ha mencionado en el test anterior, existen dos colas para los procesos que se encuentran bloqueados: una para aquellos que ejecutan un delay, y otra para aquellos que son interrumpidos por una tarea de mayor prioridad.

La idea es crear varias tareas de menor prioridad, una tarea de mediana prioridad que ejecute un cuerpo con tiempo conocido y otra de máxima prioridad que se bloquea en la cola de tiempos. De esta forma se podrá analizar el tipo de planificador. Estas tareas se pueden definir de la siguiente forma:

a) Tareas de alta prioridad que se duermen por un tiempo grande al inicio de su ejecución.
b) Tarea de media prioridad que constantemente está ejecutando un trozo de código y lo mide.
c) Tareas de baja prioridad pero ascendente, que comienzan a ejecutarse y se van bloqueando en cadena entre ellas, menos la última que será bloqueada por b.

Seguidamente, se realizan tests donde se estudian las posibles combinaciones entre las tareas anteriores, es decir:

T.b) Solo se ejecuta la tarea b.

T.ba) Se ejecuta la tarea b con un conjunto de tareas del tipo a.

T.bc) Se ejecuta b con un conjunto de tareas del tipo c.

T.bac) Se ejecutan todas las tareas a la vez, de forma que primero comiencen las del tipo a. Con esto se consiguen que todas las tareas se encuentren en la cola de delays. Seguidamente, se van levantando correlativamente las tareas del tipo c en orden ascendente de prioridad. Con esto se consigue que se bloqueen entre ellas. Finalmente, lanzamos la tarea de toma de tiempos b, la cual bloquea a la tarea c (que a su vez esta bloqueando a sus tareas iguales), y medirá continuamente su código interno.

Nota.- La idea de introducir varias tareas del tipo c y a es para identificar si existe alguna sobrecarga notable en el sistema por el aumento de tareas.

CODIGO

a)

```
delay XXX
código que consume CPU con
```

b)

```
t1:= tsc.read_tsc;
código que consume CPU con ci=X
t2:= tsc.read_tsc;
dur:= t2 – t1;
```

c)

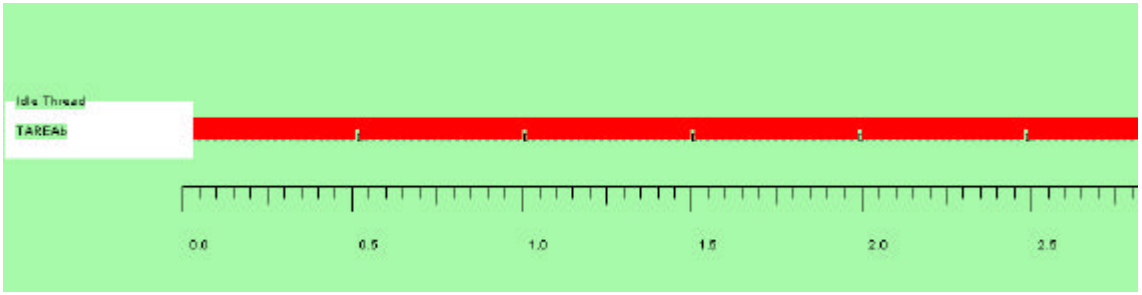
```
delay XXX
código que consume CPU
```

TIEMPOS OBTENIDOS

T.b)

Si ci= 500 µseg => dur = 500 µseg

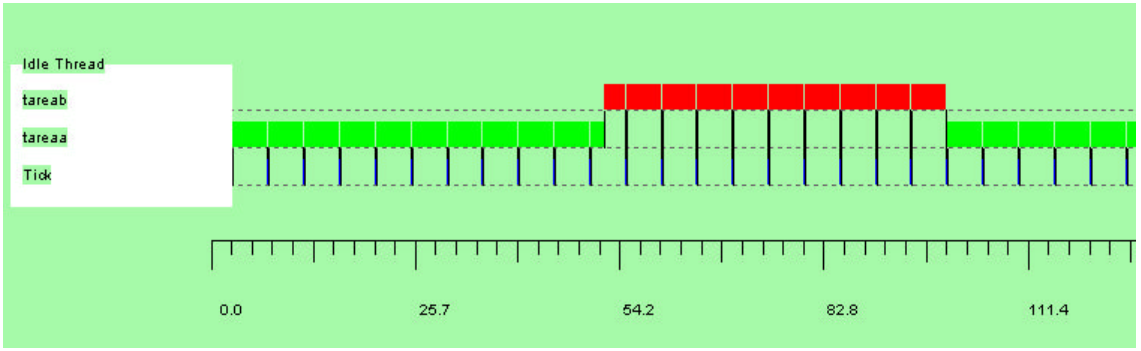
Si ci= 1000 µseg => dur= 1000 µseg



T.ba)

si ci= 500 µseg => dur = 516 µseg

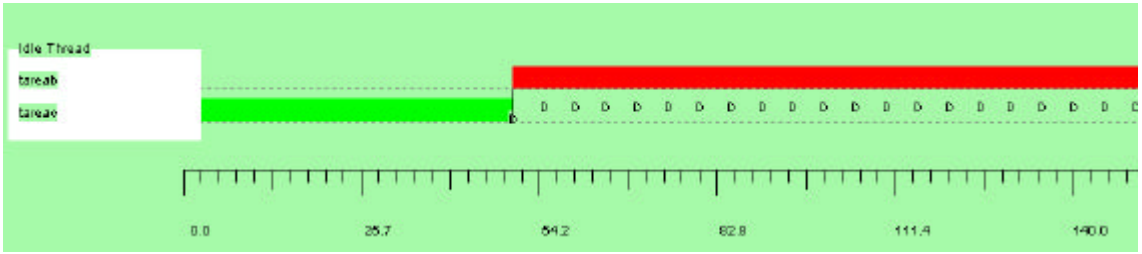
si ci= 1000 µseg => dur = 1032 µseg



T.bc)

si ci= 500 µseg => dur = 500 µseg

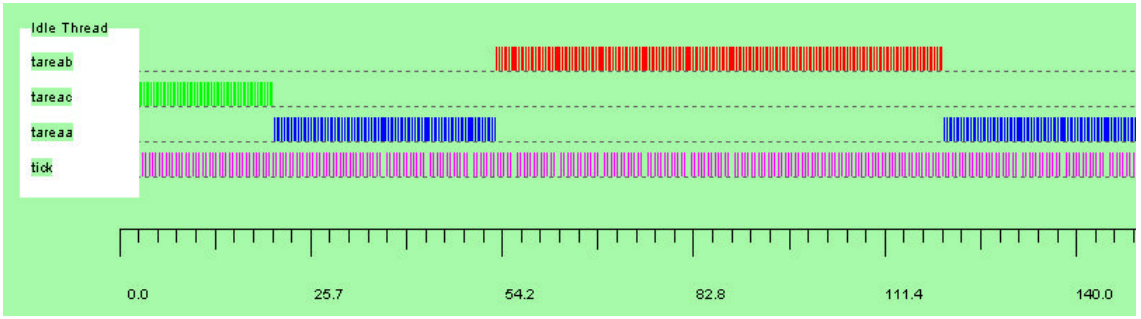
si ci= 1000 µseg => dur = 1000 µseg



T.bac)

si ci= 500 µseg => dur = 516 µseg

si ci= 1000 µseg => dur = 1032 µseg



CONCLUSION

Como se puede apreciar, cuando tenemos tareas que se bloquean en la cola de delay, aparece una sobrecarga de 16 µseg por cada 500 µseg. Esto puede ser debido a la aparición de una tarea periódica con T=500 µseg y C=16 µseg que aparece en el sistema. Dicha tarea podría ser un planificador dirigido por ticks. (cronogramas T.ba y T.bac).

Por el contrario, si se bloquea una tarea por otras de mayor prioridad, no aparece ninguna sobrecarga en el sistema. (cronograma T.bc)

Esto nos lleva a la conclusión de que en realidad, lo que se creía que era un tick, es una acción interna del procesador, la cual está asociada con el timer de las tareas que se encuentran bloqueadas. Dicha acción es la que controla las expiraciones del timer utilizado cuando se bloquean tareas.

Por lo tanto, esta operación interna hay que tenerla en cuenta a la hora de planificar, debido a que las tareas periódicas, por definición en Ada, se bloquean hasta su próxima activación una vez que completan su ejecución.

Nota.- En lo que sigue se tomará la siguiente cota superior (en tiempo de ejecución) para la acción interna del procesador:

T = 500 µseg C = 20 µseg

ya que no supone una sobrecarga muy grande en el sistema y nos dará mayor seguridad a la hora de planificar.

T.4) Tiempos de despertado de las tareas

Los tiempos de despertado de las tareas, se refieren al tiempo que tarda el sistema en llevar las tareas que se encuentran en la cola de bloqueados a la cola de ready.

En nuestro caso consistirá en pasar las tareas desde la cola asociada a delay hacia la cola de ready.

La idea es bloquear un conjunto de tareas hasta una hora fija y medir el desfase respecto a la hora real a la que se levantan. Para ello, se realizarán los siguientes 4 tests, de acuerdo con las horas de activación y las prioridades de las tareas y variando el número de tareas:

T.HP1) Misma hora, misma prioridad

T.HP2) Misma hora, distinta prioridad

T.HP3) Distinta hora, misma prioridad

T.HP4) Distinta hora, distinta prioridad

CODIGO

El código para los cuatro tests será el mismo, solo cambiarán en los valores asignados de prioridad y tiempos de activación.

```
Next_time:= clock + periodo;

For i in 1 .. N loop

        T_real := tsc.read_tsc;

        Delay until next_time;

        Next_time:= next_time + periodo;

        Dur:= T_real – next_time;

End loop;
```

TIEMPOS OBTENIDOS

T.HP1)

Nº tareas = 1 T_activación = (72, 74) µs

Nº tareas = 4 T_activación = {200, 270, 330, 390} µs

Nº tareas = 16 T_activación = {719, 784, . . . 1655, 1740}µs

T.HP2)

Nº tareas = 1 T_activación = (72 , 74) µs

Nº tareas = 2 T_activación = {107, 178} µs

Nº tareas = 3 T_activación = {142, 216, 287} µs

Nº tareas = 10 T_activaciòn = {350, 425, 495 . . .} µs

Nº tareas = 16 T_activaciòn = {515 , 590, 680 . . .} µs

Nº tareas = 26 T_activación = {880, 950, 1024 . . .} µs

T.HP3)

Nº tareas = 1 T_activación = (72 , 74) µs

Nº tareas = 4 T_activación = {76 , 84, 84, 84} µs

Nº tareas = 16 T_activación = {100, 100, . . .} µs

T.HP4)

Nº tareas = 1 T_activación = (72 , 74) µs

Nº tareas = 2 T_activación = {75 , 80} µs

Nº tareas = 3 T_activación = {75, 80, 80} µs

Nº tareas = 4 T_activación = {80, 80, 80,80} µs

Nº tareas = 5 T_activación = {80, 80, 80, 80} µs

Nº tareas = 6 T_activación = {100, 2 85 } µs

Nº tareas = 7 T_activación = {100, 100, 3 85} µs

Nº tareas = 8 T_activación = {1 100, 4 85} µs

Nº tareas = 9 T_activación = {1 100, 5 85} µs

Nº tareas = 10 T_activaciòn = {1 100, 6 85} µs

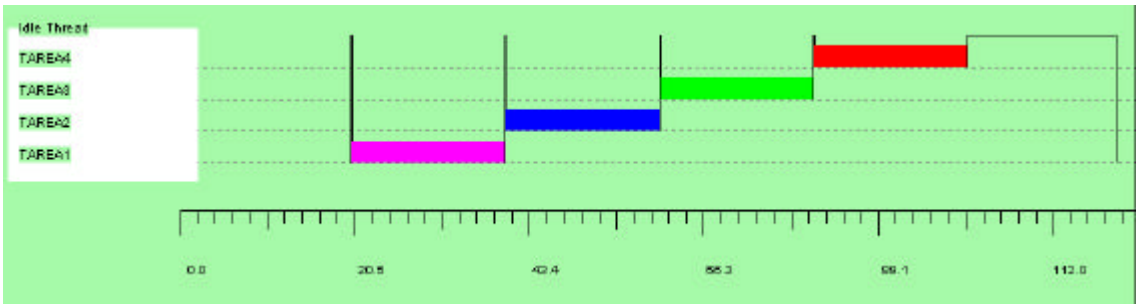
Nº tareas = 11 T_activación = {1 100, 7 85} µs

Nº tareas = 12 T_activación = {1 100, 8 85, 12 110} µs

Nº tareas = 13 T_activación = {1 100, 9 85, 12 110} µs

Nº tareas = 14	T_activación = {1 100,10 85, 12 110} µs
Nº tareas = 15 . . 21	T_activación = {1 110} µs
Nº tareas = 22	T_activaciòn = {180, 2 110, 17 120, 22 150} µs
Nº tareas = 23	T_activación = {180, 4 110, 17 120, 22 150} µs
Nº tareas = 24	T_activación = {180, 4 110, 18 120, 22 150} µs
Nº tareas = 25	T_activación = {180, 4 110, 18 120, 22 150} µs
Nº tareas = 26	T_activación = {180, 5 110, 21 120, 24 150} µs

CRONOGRAMA



CONCLUSIONES

Como se puede apreciar, el tiempo de cola aumenta a medida que se incrementa el número de tareas que tienen que activarse a la misma hora. Si se tienen que activar a horas diferentes, entonces se mantiene constante.

El que las prioridades sean diferentes, no afecta considerablemente a los resultados.

Para nosotros será más interesante analizar el caso en el que las tareas tienen distinta prioridad y diferente tiempo de activación, ya que estamos utilizando la expresión referente a los tiempos de respuesta de tareas de distintas prioridades y modelando el planificador por eventos. Por lo tanto nos centraremos en los datos obtenidos en el test T.HP4.

Nota.- El caso en el que las tareas se levanten a la misma hora (T.HP2) está contenido en T.HP4, pero carece de valor cuando estudiamos los peores tiempos de respuesta para un conjunto de tareas.

Por lo tanto, una buena cota superior para la transición realizada por el planificador, cuando pasa tareas desde la cola de suspendidos a la de preparados según el número de tareas que posea la aplicación, es:

nº tareas aplicación	**Tiempo**
1 5	80 µs
6 11	100 µs
12 21	110 µs
22 26	150 µs

Es decir, los valores de esta tabla nos servirán para poder definir más exactamente el tiempo de ejecución de $C_{timer,}$ cuando estamos realizando el análisis de planificabilidad por tiempo de respuesta con planificación basada en eventos.

P) Planificación de procesos

Todos los tests realizados hasta ahora han ido enfocados a obtener los parámetros temporales contenidos en la siguiente expresión:

$$R_i = C_i + 2\,C_{SW} + B_i + \sum_{j \in Hp(i)}^{T} \left[(R_i + J_j) / T_j \right] * (C_j + 2\,C_{SW}) + \left[C_i / T_{Itimer} \right] * C_{Itimer} + \sum_{j=1}^{T} \left[(R_i + J_j) / T_j \right] * C_{timer}$$

Esta expresión ha sufrido una pequeña variación con respecto a la original, para poder así adecuarla al modelo de planificación de tareas periódicas basada en eventos de MaRTE O.S.. La modificación ha sido:

- Añadir la interrupción originada por la acción interna del procesador, es decir:

$$C_{Itimer} = 20 \ \mu\text{segundos}; \qquad T_{Itimer} = 500 \ \mu\text{segundos};$$

$$\left[C_i / T_{Itimer} \right] * C_{Itimer}$$

En los tests que restan se aplicará dicha expresión, para comprobar si un conjunto de tareas periódicas es o no planificable, según los tiempos de ejecución y periodos de activación asignados a cada tarea.

P.1) Planificación de procesos independientes

Ahora se van a poner a prueba los parámetros obtenidos referentes al kernel obtenidos hasta el momento.

La idea es planificar un conjunto de tareas independientes entre sí, por medio del análisis de tiempo de respuesta.

Para ello, se realizarán dos tests, cada uno con diferentes características temporales, de forma que uno sea planificable y el otro no.

P.1.1) Conjunto no planificable de tareas.

TEST

Consistirá en ejecutar el siguiente conjunto de tareas periódicas:

TAREA	PERIODO	T_EJEC.	PRIORIDAD
1	10	4.0	1
2	4	2.0	2
3	2	0.5	3

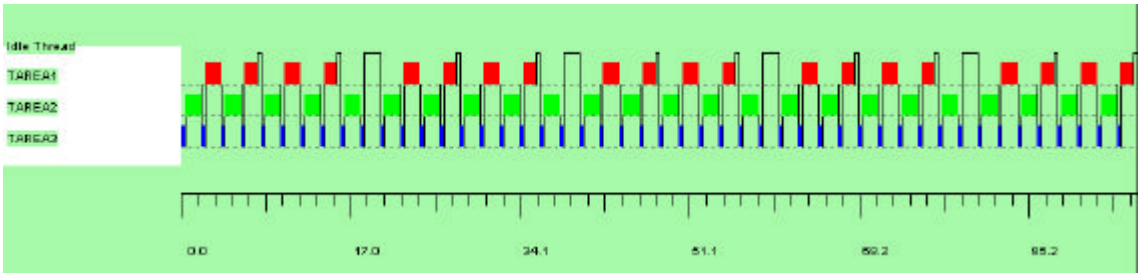
CODIGO

El código para todas las tareas es el mismo:

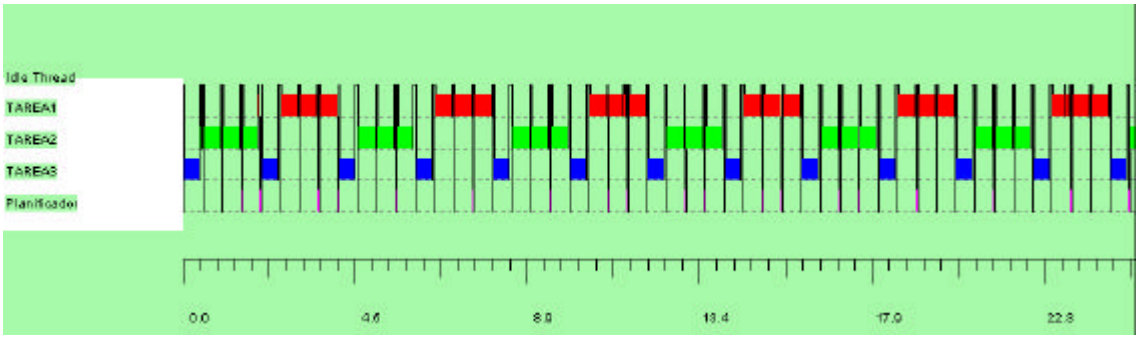
```
Id := 'identificador de la tarea'
Next_time:= clock + hora_relativa;

For I in 1 .. N loop
        If id = 1 then
                Select
                        Delay until next_time;
                        Put(" Error Tarea 1");
                Then abort
                        Acción (t_ejec_id);
                End select;
        else
                Acción (t_ejec_id);
                Delay until next_time;
        End if;
        Next_time := next_time + periodo;
End loop;
```

CONCLUSION

Si no existieran, cambios de contexto, interrupciones del sistema, etc, entonces el conjunto de tareas sería planificable y tendría el siguiente aspecto:



Pero como se ha visto hasta ahora, esto no ocurre, por lo tanto la planificación real para los datos temporales anteriores es:



Donde se observan ejecuciones fuera de periodo en la tarea_1.

Por lo tanto el conjunto de tareas no es planificable para los parámetros temporales de las tareas.

P.1.2) Conjunto planificable de tareas.

TEST

Consistirá en el siguiente conjunto de tareas periódicas:

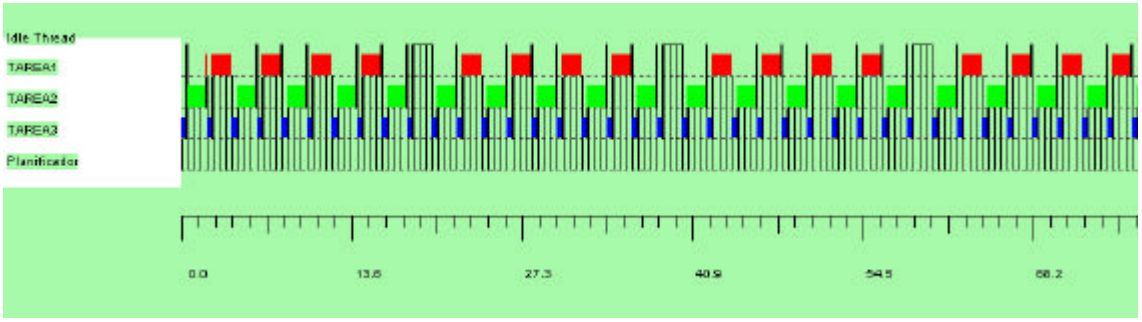
TAREA	PERIODO	T_EJEC.	PRIORIDAD
1	10	2.0	1
2	4	1.0	2
3	2	0.5	3

CODIGO

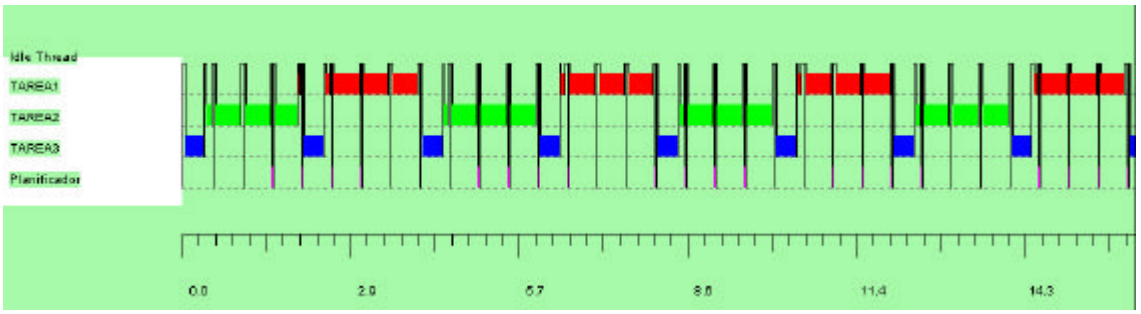
El código para todas las tareas es el mismo:

```
Id := 'identificador de la tarea'
Next_time:= clock + hora_relativa;

For I in 1 .. N loop
        If id = 1 then
                Select
                        Delay until next_time;
                        Put(" Error Tarea 1");
                Then abort
                        Acción (t_ejec_id);
                End select;
        else
                Acción (t_ejec_id);
                Delay until next_time;
        End if;
        Next_time := next_time + periodo;
End loop;
```

CRONOGRAMA



Ampliación de los primeros 10 milisegundos de ejecución:



CONCLUSION

Como se puede apreciar, el conjunto de tareas ahora si es planificable.

Por lo tanto, este conjunto de tareas periódica es planificable por el modelo de análisis de tiempo de respuesta, ordenadas según el algoritmo de asignación monótona de prioridades y expulsivo. Tomando el modelando de planificador por eventos y utilizando los datos obtenidos en los tests realizados.

P.2) Planificación de procesos que se comunican

En este test, se tratará un conjunto de tareas periódicas que acceden a dos tipos protegidos. Esto podría ser considerado como una comunicación entre tareas, debido a que en la programación de aplicaciones en tiempo real se utilizan dichas variables para la compartir datos.

A partir de este test, se podrá obtener el tipo de protocolo utilizado por MaRTE OS a la hora de planificar tareas que utilizan tipos protegidos.

TEST

Para la realización de dicho test, se tomarán los siguientes parámetros para las tareas y las variables protegidas:

TAREA	PRIORIDAD	SECUENCIA DE EJECUCION	T. ACTIVACION
τ_1	4	EEABE	4
τ_2	3	EBBE	2
τ_3	2	EE	2
τ_4	1	EAAAAE	0

RECURSO	TECHO DE PRIORIDAD
A	4
B	4

Se van ha realizar dos test (A y B), cuya única diferencia es el tiempo de ejecución. A del orden de segundos y B del orden de milisegundos.

CODIGO

```
Next_time:= next_time + clock + activacion(id);

For I in 1 .. N loop
        Delay until next_time;

        If id = 4 then
                Acción (t_ejec_1);
                Obj_prot_A.entrada(t_ejec_4);
                Acción (t_ejec_1);

        Elsif id = 3 then
                Acción (t_ejec_1);
                Acción (t_ejec_1);

        Elsif id = 2 then
                Acción (t_ejec_1);
                Obj_prot_B.entrada(t_ejec_2);
                Acción (t_ejec_1);

        Elsif id = 1 then
                Acción (t_ejec_1);
                Acción (t_ejec_1);
                Obj_prot_A.entrada(t_ejec_1);
                Obj_prot_B.entrada(t_ejec_1);
                Acción (t_ejec_1);

        End if;

        Next_time := next_time + periodo;
End loop;
```

TIEMPOS OBTENIDOS

A) Tiempos de ejecución:

T_ejec_1 := 1 segundo

T_ejec_2 := 2 segundos

T_ejec_4 := 4 segundos

Tiempos de respuesta:

Tarea1: H_inicio = + 1’000_000 segundos

T_ejec. = 5’385_000 segundos

Tarea2: H_inicio = + 8’135_000 segundos

T_ejec. = 4’140_000 segundos

Tarea3: H_inicio = + 12’276_000 segundos

T_ejec. = 2’070_000 segundos

Tarea4: H_inicio = + 0’000_086 segundos

T_ejec. = 17’380_000 segundos

B) Tiempos de ejecución:

T_ejec_1 := 1 msegundos

T_ejec_2 := 2 msegundos

T_ejec_4 := 4 msegundos

Tiempos de respuesta:

Tarea1: H_inicio = + 0’001_000 segundos

T_ejec. = 0’005_292 segundos

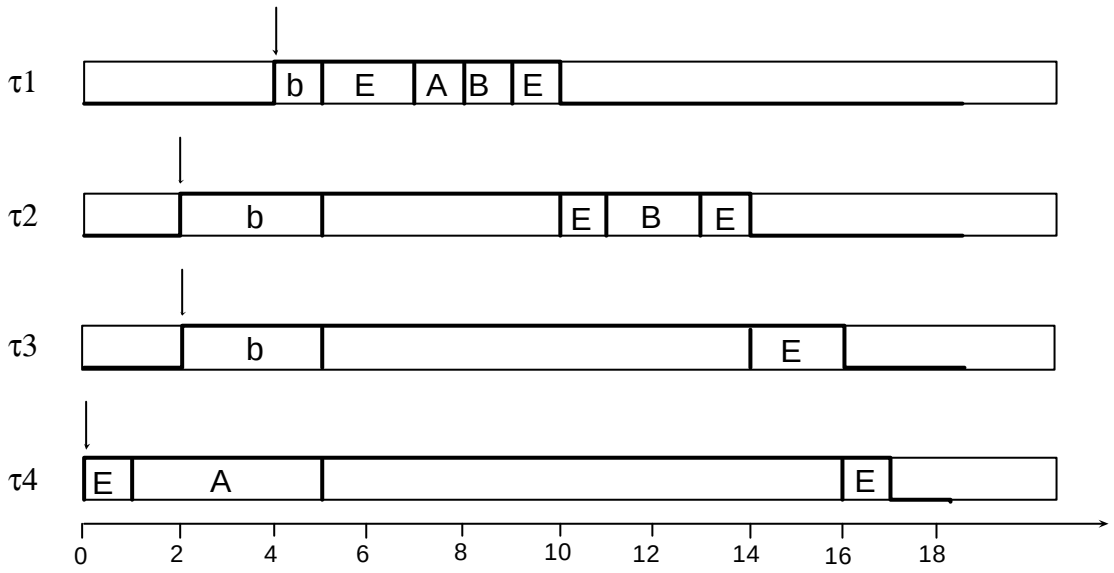
Tarea2: H_inicio = + 0’002_000 segundos

T_ejec. = 0’004_380 segundos

Tarea3:	H_inicio =	+ 0’013_490	segundos
	T_ejec. =	0’002_250	segundos
Tarea4:	H_inicio =	+ 0’000_080	segundos
	T_ejec. =	0’019_000	segundos

Nota.- Los signos (+) indican la hora de comienzo de las tareas contando desde el instante de activación teórico.

CRONOGRAMA



CONCLUSION

Se puede concluir que el protocolo utilizado por MaRTE OS, es el ICPP (Inmediate Ceiling Protocol o Protocolo de Techo Inmediato), el cual es idóneo a la hora de acotar y medir el tiempo transcurrido por la ocurrencia de una inversión de prioridad, en un sistema de tiempo real.

En el test A, donde el tiempo base es del orden de segundos, se puede apreciar que se adapta bastante bien.

En B, donde el tiempo basa es del orden de milisegundos, se aprecia que no es tan real. Esto es debido a varios motivos:

- a la acción de una tarea periódica asociada al timer del planificador, cuyo periodo es de 500 µseg. y su Ci es de 20 µseg. Por lo tanto, con medidas tan pequeñas el efecto aquí si se hace notar.

- Los tiempos de activación de las tareas, también son bastante relevantes.

Por lo tanto, MaRTE OS trabaja bien con la comunicación entre tareas utilizando tipos protegidos, cuando los parámetros temporales son del orden de segundos.

4.3 CONCLUSIONES GENERALES SOBRE MaRTE O.S.

MaRTE O.S. (Minimal Real-Time Operating System for Embedded Aplicatios) es un kernel de tiempo real para aplicaciones embebidas que sigue el subconjunto POSIX.13, además añade las extensiones POSIX.1d y POSIX.1j.

La principal ventaja de este sistema, es que soporta aplicaciones escritas en Ada y C usando los compiladores G.N.U. gnat y gcc.

Debido a que el entorno de ejecución de Gnat soporta completamente los perfiles POSIX (citados anteriormente), en MaRTE OS es posible crear aplicaciones que sean portables para arquitecturas de PC diferentes, siempre y cuando no utilicen sistemas de ficheros (restricción impuesta en POSIX.13-PSE5.0).

Debido a que MaRTE OS trabaja en un entorno de desarrollo HOST/TARGET, entonces será posible crear aplicaciones empotradas de tiempo real utilizando un 'PC vacío'.

La instalación del entorno de desarrollo no es difícil, lo único que hay que tener en cuenta es utilizar las versiones de los paquetes especificados en el proceso de instalación.

Referente a los parámetros de planificación obtenidos en esta evaluación (para un PC Intel 120), se puede concluir que MaRTE OS:

- Tiene un comportamiento temporal predecible.
- Se pueden ejecutar (de forma segura) aplicaciones de tiempo real con tiempos de ejecución del orden de milisegundos.
- El planificador que utiliza es por eventos, y utiliza estructuras para los procesos bastante eficientes a la hora de ejecutar una aplicación.

Como apunte final, MaRTE OS viene siendo desarrollado por el Departamento de Electrónica y Computadores de la Universidad de Cantabria bajo licencia G.N.U, por lo tanto puede ser descargado gratuitamente desde http://marte.unican.es con fines académicos para el desarrollo de nuevas herramientas SW para ser añadidas a este eficiente microkernel.

Parámetros temporales obtenidos en la evaluación técnica de MaRTE O.S. sobre un TARGET Intel Pentium 120 MHz:

PARAMETROS MEDIDOS	TIEMPOS OBTENIDOS
Tiempo de lectura del reloj del sistema	3 µsegundos
Tiempo de ejecución de la instrucción delay until	40 µsegundos
Cota inferior para el intervalo de la instr. delay until	155 µsegundos
Cambio de contexto entre tareas	50 µsegundos
Jitter de activación de las tareas	12 µsegundos
Tiempo de despertado de las tareas (según número de tareas que se encuentran en el sistema)	1 5 tareas = 80 µs 6 11 tareas = 100 µs 12 21 tareas = 110 µs 21 26 tareas = 150 µs

Con estos parámetros y utilizando la expresión para el análisis de tiempos de respuesta se podrá analizar la planificabilidad de un conjunto de tareas escritas en Ada95 y compiladas bajo MaRTE O.S. para su posterior ejecución.

$$R_i = C_i + 2C_{SW} + B_i + \sum_{j \in Hp(i)}^{T} \left[(R_i + J_j) / T_j \right] * (C_j + 2C_{SW}) + \left[C_i / T_{Itimer} \right] * C_{Itimer} + \sum_{j=1}^{T} \left[(R_i + J_j) / T_j \right] * C_{timer}$$

CAPITULO 5. Conclusiones

5.1 CONCLUSIONES SOBRE EL TRABAJO REALIZADO

El propósito de este P.F.C. ha consistido en diseñar un banco de pruebas, que nos permitiera determinar y medir los parámetros temporales más representativos de MaRTE OS a la hora de crear aplicaciones de tiempo real.

Después de instalar y analizar MaRTE OS, se han llegado a las siguientes conclusiones:

La instalación debe hacerse utilizando las versiones 'exactas' de los paquetes especificados en el proceso de instalación, ya que se ha intentado con otras versiones y no funcionan bien o dan problemas entre versiones.

Hay que recompilar el kernel para que tenga mas recursos de sistema, ya que la instalación inicial solamente soporta la ejecución de doce tareas.

Los tests han sido realizados bajo el lenguaje Ada 95, por lo tanto son válidos para aplicaciones que sean realizadas en dicho lenguaje. Los tiempos obtenidos son del orden de microsegundos.

Las aplicaciones escritas siguiendo el estándar Posix 5 no han sido analizadas. Esto ha sido debido en parte, a que es otro interfaz que puede ser utilizado independientemente del resto y tendría que ser estudiado conforme a las especificaciones propuestas según en dicho estándar.

Los resultados de las pruebas realizadas dependerán sobre que tipo de Target que se utilice (ya sea PC Intel 486, Familia P5 ó Familia P6), debido a que los relojes y timers utilizados en el sistema cambian, según la arquitectura interna del microprocesador.

Las pruebas realizadas sobre un conjunto de tareas (escogido previamente de forma que sobrecargase lo más posible al procesador) han sido satisfactorias, ya que se ha podido determinar si un conjunto de tareas si es planificable o no atendiendo a los valores temporales específicos de cada una.

El conjunto de tests seleccionado podría haber sido más amplio, pero el escogido en este proyecto es suficiente para la obtención de los parámetros temporales necesarios al aplicar la expresión de planificación por tiempos de respuesta utiliza.

En definitiva, MaRTE O.S. es un buen entorno de desarrollo para aplicaciones de tiempo real que cumple el interfaz mínimo de Posix 13. También es muy indicado para la programación de S.T.R. con tiempos del orden de milisegundos. Además está desarrollado bajo licencia G.N.U., la cual permite la libre adquisición y desarrollo del producto con fines académicos y de investigación.

CAPITULO 6. Apéndices

6.1 HERRAMIENTAS SW Y HW UTILIZADAS

1.- Softaware

- S.O. Linux Red Hat 6.0
- Gnat 3.13
- Potocolo bootp 2.4
- Sistema de ficheros NFS
- MaRTE O.S.

2.- Hardware

HOST

- AMD K6-2 500 MHz
- Tarjeta Ethernet Real Teck
- Monitor
- Disco duro
- Memoria RAM

TARGET

- Intel Pentium 120
- Disquetera 1.44 Mbytes
- Tarjeta Ethernet Real Teck
- Memoria RAM

6.2 NOTACION UTILIZADA EN EL PROYECTO

SW	Software
HW	Hardware
S.O.	Sistema Operativo
S.O.T.R.	Sistema Operativo de Tiempo Real
S.0.P.G.	Sistema Operativo de Propósito General
N.F.S.	Network File System
S.T.R.	Sistema de Tiempo Real
P.O.S.I.X	Portable Operating System Interface basado en uniX
A.P.I.	Aplication Program Interface
G.N.U.	General Public License
T.S.C.	Timer Stamp Counter
Deadline	Límite de tiempo definido para una tarea

Bibliografía

Capítulo 2

2.1 Capítulo 1. ALAN BURNS & ANDY WELLINGS. ”Real-Time Systems and Programing Languages”. Addison-Wesley.

2.3 http://www.jap.upm.es Web personal de apuntes sobre S.T.R. de Juan Antonio de la Puente.

2.4 http://marteos.unican.es POSIX de Tiempo Real. Michel González Harbour.

2.5 http://www.dedicated-systems.com

Capítulo 3

3.2 BILL BALL & DAVID PITTS. “Red Hat Linux a fondo”. Anaya.

http://marteos.unican.es User-Guide.

Bootpd. Manuales de Linux

3.3 http://marteos.unican.es MaRTE OS: An Ada Kernel for Real-Time Embedded Applications. Mario Aldea Rivas and Michael González Harbour.

Capítulo 4

4.1 Capítulo 13. ALAN BURNS & ANDY WELLINGS. ”Real-Time Systems and Programing Languages”. Addison-Wesley.