ESTUDIO DE MIGRACIÓN DE LA PLATAFORMA TIEMPO REAL PARA EL COMPUTADOR PC104

Br. Carlos Mejia
Tutor: Prof. Leandro León
Cotutor: Prof. Addison Ríos

PROYECTO DE GRADO PRESENTADO ANTE LA ILUSTRE UNIVERSIDAD DE LOS ANDES COMO REQUISITO FINAL PARA OPTAR AL TITULO DE INGENIERO DE SISTEMAS

Mérida, Venezuela
Octubre 2006

CALIFICACIÓN DE
PROYECTO DE GRADO

Los suscritos, Miembros del Jurado designado para evaluar el Proyecto de Grado titulado "**Estudio de Migración de la Plataforma Tiempo Real Para el Computador PC104**" realizado por Br. Mejia Torres Carlos Miguel, C.I. Nº 15.135.906, presentado como requisito parcial para la obtención del grado de Ingeniero de Sistemas, Han resuelto, de acuerdo a lo pautado en el Artículo 13 del Reglamento de Proyecto de Grado, colocar la calificación de

CALIFICACIÓN: _____

Fecha: Octubre de 2006

OBSERVACIONES: _____

_____

_____

Profesor Guia: _____
Prof. Leandro León

Cotutor: _____
Prof. Addison Ríos

Jurado: _____
Prof. Wladimir Rodríguez

_____
Prof. Víctor Bravo

Índice

Índice de Figuras

Índice de Tablas

Agradecimientos

"…A merced de un ser omnisciente, he asomado mi existencia hasta aquí, tratando de convencerme que ha sido por voluntad propia, motivado por alcanzar los aires de independencia que pueden oírse a lo lejos, buscando respuestas a eso sueños que no puedo recordar, desesperado por saciar ese hambre de conocimiento que me mantiene despierto por las noches; todo un manojo de razones por las cuales dejar a las personas que me han apoyado todo mi vida pero que nunca me han abandonado; mis logros solo son un pequeño suspiro de los que ellos pueden hacer, se las arreglaron para estar allí cuando mas los necesite, una madre en forma de luna que le quito ese sabor a soledad que tienen las noches, un padre en forma de sol capaz de desplazar ese frió que guardan las ciudades desconocidas. Internarse todo los días en pasillos de guerra, donde tus conversaciones son parte del ruido matutino, te hacen victima de la absurda amistad en los lugares menos pensados, como rincones llenos de historia y paredes pintadas de recuerdos, hasta que conoces a un grupo de personas que desean cambiar lo que eres y borrar lo que has sido, cuando te das cuentas quizás no te reconozcas en el espejo y es allí donde comenzaras a buscar le esencia de las cosas dándote cuenta de lo que en verdad es importante; finalmente hallaras a ese grupo de personas sin nombres que te aceptan por lo que eres y que te valoran por lo que puedes llegar a ser; cuando todo esta a punto de terminar surgen oportunidades que te permiten experimentar la independencia que andabas buscando y de saciar por raciones tu moribundo conocimiento hasta que eres adicto al destino. Si lo piensas no solo depende de ti llegar hasta aquí, es el resultado de la combinación de la motivación que generan los padres, mas las ganas que tengas de seguir adelante, sumando a las personas que te rodean en tu DIA a DIA, es decir, que el ego debe de salir de paseo porque tus logros son el producto de la colaboración de mucha gente que merecen tu agradecimiento. En mi caso las personas que lograron que esto fuese posible son mis padres por apoyarme siempre, mi familia por creer en mi, mis amigos por no olvidarme, mi tutor por brindarme su confianza, y mi grupo de trabajo en el proyecto 2005000170 quienes me brindaron las condiciones y herramientas aptas para lograr llegar a donde estoy en este momento. MUCHAS GRACIAS!!!...."

Resumen

En la industria lo sistemas empotrados cada vez son mas utilizados, gracias a sus ventajas físicas y funcionales, como es el caso del PC104 cuyo bajo consumo de energía, fácil integración y adecuado rendimiento en situaciones adversas le han dado un puesto en los actuales sistemas de control, supervisión o monitorización. Normalmente este sistema empotrado (PC104) funciona bajo una plataforma de tiempo real comercial, es decir, el software que manipula este sistema es licenciado y sus costos pueden llegar a ser muy elevados.

En este estudio se desarrolla una nueva plataforma de tiempo real para el PC104 basada en software libre, construyéndose un sistema operativo capaz de manipular las operaciones del PC y apto para soportar exigencias de tiempo real. El sistema se ha construido utilizando un kernel de Linux básico y una interfase para aplicaciones de tiempo real llamada RTAI, la cual permite desarrollar aplicaciones con estrictas exigencias de tiempo sobre cualquier sistema Linux. Para construir el sistema primero es necesario conocer a fondo la arquitectura del PC104, por lo que se hace uso de los manuales del PC y así poder conocer ciertas características importantes, como los tipos de dispositivos de entrada y salida, módulos de conexión de red, tipo y frecuencia del procesador, buses de comunicación, entre otros. Teniendo en cuenta estas características se puede configurar el nuevo kernel del sistema de una forma optima, el cual necesita primero ser parchado para soportar las aplicaciones desarrolladas sobre RTAI, este parche es proporcionado por los mismos desarrolladores de RTAI. El nuevo sistema que funcionara sobre el PC104 se ha construido desde cero, es decir, se tiene un total dominio de todos los paquetes y herramientas que conforman el sistema operativo.

Luego de haber terminado el sistema, se han realizado ciertas aplicaciones que demuestran el correcto funcionamiento de RTAI, además de algunas de sus características más relevantes y la posibilidad de poder utilizar el PC104 como un controlador programable capaz de cumplir con tareas de supervisión y control sobre distintos procesos físicos.

Descriptores: Sistemas Empotrados, PC104, Sistemas de Tiempo Real, Micro-Kernel, Control de Procesos.

Capítulo 1

Introducción

Las técnicas de control han ido evolucionando y avanzando a medida que cambian los procesos industriales, y así poder adaptarse a las nuevas exigencias de rendimiento, comunicación, costos, integración hasta climatológicas, lo cual a llevado a cabo la construcción e implementación de sistema empotrados, ya que estos tienen un menor consumo de energía, son económicos y pueden armarse por módulos según sean las características del proceso. Estos sistemas tienen casi las mismas funciones que un computador personal, excepto que son construidos para situaciones muy específicas.

El PC104 es un estándar de estos sistemas empotrados que define la placa base (PC) y el bus del sistema (ISA, PCI), la arquitectura de la placa base no es la típica placa de circuitos integrados en el que van insertados los componentes, en lugar de eso, los componentes se encuentran en módulos que son apilados unos encima de otros. El PC104 está diseñado para aplicaciones específicas, como adquisición de datos o sistemas de control industrial.

Para operar el PC104 es necesario un sistema operativo capaz de soportar exigencias de tiempo real y que sea confiable ante la manipulación de datos, dando respuestas en los intervalos de tiempo establecidos. Estos sistemas de tiempo real son predecibles (deterministas) para poder cumplir con las exigencias de tiempo, además no utilizan mucha memoria y cualquier evento en el soporte físico puede hacer que se ejecute una tarea. En el mercado existe cierta variedad de estos sistemas de tiempo real, pero en su

mayoría son algo costosos y cualquier necesidad en particular por parte de ellos requiere un costo adicional, por lo que es mas útil recurrir a un sistema de tiempo real basado en software libre, lo cual nos permite una total manipulación del sistema y así poder ajustarlo a nuestras necesidades.

En los primeros capítulos se presentan los fundamentos teóricos respecto a los sistemas empotrados, el PC104, y los sistema operativos de tiempo real, en el penúltimo capítulo se presenta la construcción de un sistema de tiempo real para el PC104 y en el último capítulo se desarrollan algunas aplicaciones sobre el nuevo sistema, de tal forma que se puedan verificar sus facultades.

Capítulo 2

Sistemas Empotrados

En este capítulo se presentan las principales características de los sistemas empotrados, así como su arquitectura y algunas de las aplicaciones donde se pueden utilizar estos sistemas.

2.1 Introducción

En los procesos de control, es necesario contar con un dispositivo que este en contacto con las variables del proceso así como con los demás dispositivos de campo; Este dispositivo debe de ser capaz de soportar adversidades naturales, como altas temperaturas y humedad, además de tener un tamaño reducido y bajo consumo de energía; en consecuencia de estas necesidades físicas y de las exigencias de tiempo real, surgen los sistemas empotrados, capaces de realizar tareas dedicadas muy especificas y con un comportamiento determinista.

Los sistemas empotrados ocupan ese espacio vació que dejan los computadores personales en procesos de control en zonas de alto riesgo o lugares de espacio limitado por el tamaño o peso del dispositivo; la inclusión de un computador como herramienta de control de procesos, le da fuerza al desarrollo e implementación de sistemas empotrados para cumplir con clásicas o nuevas estrategias de control.

Los primeros equipos empotrados que se desarrollaron fueron elaborados por IBM en los años 1980.

2.2 Componentes de un Sistema Empotrado

En la parte central se encuentra el microprocesador, microcontrolador, DSP, etc. Es decir el CPU o unidad que aporta inteligencia al sistema. Según el sistema puede incluir memoria interna o externa, un micro con arquitectura especifica según requisitos. [21]

La comunicación adquiere gran importancia en los sistemas empotrados. Lo normal es que el sistema pueda comunicarse mediante interfaces estándar de cable o inalámbricas. Así un SE normalmente incorporará puertos de comunicaciones del tipo RS232, RS485, SPI, I²C, CAN, USB, IP, WiFi, GSM, GPRS, DSRC, etc. [21]

El subsistema de presentación suele ser una pantalla gráfica, táctil, LCD, alfanumérico, etc.

Denominamos actuadores a los posibles elementos electrónicos que el sistema se encarga de controlar. Puede ser un motor eléctrico, un conmutador tipo relé etc. El más habitual puede ser una salida de señal PWM para control de la velocidad en motores de corriente continua. [21]

El módulo de E/S analógicas y digitales suele emplearse para digitalizar señales analógicas procedentes de sensores, activar diodos LED, reconocer el estado abierto cerrado de un conmutador o pulsador, etc.

El módulo de reloj es el encargado de generar las diferentes señales de reloj a partir de un único oscilador principal. El tipo de oscilador es importante por varios aspectos: por la frecuencia necesaria, por la estabilidad necesaria y por el consumo de corriente requerido. El oscilador con mejores características en cuanto a estabilidad y coste son los basados en resonador de cristal de cuarzo, mientras que los que requieren menor

consumo son los RC. Mediante sistemas PLL se obtienen otras frecuencias con la misma estabilidad que el oscilador patrón. [21]

El módulo de energía se encarga de generar las diferentes tensiones y corrientes necesarias para alimentar los diferentes circuitos del SE. Usualmente se trabaja con un rango de posibles tensiones de entrada que mediante conversores ac/dc o dc/dc se obtienen las diferentes tensiones necesarias para alimentar los diversos componentes activos del circuito.

Además de los conversores ac/dc y dc/dc, otros módulos típicos, filtros, circuitos integrados supervisores de alimentación, etc. El consumo de energía puede ser determinante en el desarrollo de algunos SE, que necesariamente se alimentan con baterías es imposible su sustitución, con lo que la vida del SE suele ser la vida de las baterías.

2.3 Microprocesadores y Sistemas Empotrados

Un microprocesador es una implementación en forma de circuito integrado (IC) de la Unidad Central de Proceso CPU de un computador. Frecuentemente nos referimos a un microprocesador como simplemente “CPU”, y la parte de un sistema que contiene al microprocesador se denomina subsistema de CPU. Los microprocesadores varían en consumo de potencia, complejidad y coste. Los hay de unos pocos miles de transistores y con coste inferior a 2 euros (en producción masiva) hasta de más de cinco millones de transistores que cuestan más de 600 euros. [21]

Los subsistemas de entrada/salida y memoria pueden ser combinados con un subsistema de CPU para formar un computador o sistema empotrado completo. Estos subsistemas se interconectan mediante los buses del sistema (formados a su vez por el bus de control, el bus de direcciones y el bus de datos).

El subsistema de entrada acepta datos del exterior para ser procesados mientras que el subsistema de salida transfiere los resultados hacia el exterior. Lo más habitual es que haya varios subsistemas de entrada y varios de salida. A estos subsistemas se les reconoce habitualmente como periféricos de E/S. [21]

El subsistema de memoria almacena las instrucciones que controlan el funcionamiento del sistema. Estas instrucciones comprenden el programa que ejecuta el sistema. La memoria también almacena varios tipos de datos: datos de entrada que aún no han sido procesados, resultados intermedios del procesado y resultados finales en espera de salida al exterior.

Es importante darse cuenta de que los subsistemas estructuran a un sistema según funcionalidades. La subdivisión física de un sistema, en términos de circuitos integrados o placas de circuito impreso (PCBs) puede y es normalmente diferente. Un solo circuito integrado (IC) puede proporcionar múltiples funciones, tales como memoria y entrada/salida.

Un microcontrolador (MCU) es un IC que incluye una CPU, memoria y circuitos de E/S. Entre los subsistemas de E/S que incluyen los microcontroladores se encuentran los temporizadores, los convertidores analógico a digital (ADC) y digital a analógico (DAC) y los canales de comunicaciones serie. Estos subsistemas de E/S se suelen optimizar para aplicaciones específicas (por ejemplo audio, video, procesos industriales, comunicaciones, etc.). [21]

Hay que señalar que las líneas reales de distinción entre microprocesador, microcontrolador y microcomputador en un solo chip están difusas, y se denominan en ocasiones de manera indistinta unos y otros.

En general, un SE consiste en un sistema con microprocesador cuyo hardware y software están específicamente diseñados y optimizados para resolver un problema concreto eficientemente. Normalmente un SE interactúa continuamente con el entorno

para vigilar o controlar algún proceso mediante una serie de sensores. Su hardware se diseña normalmente a nivel de chips, o de interconexión de PCBs, buscando la mínima circuitería y el menor tamaño para una aplicación particular.

Otra alternativa consiste en el diseño a nivel de PCBs consistente en el ensamblado de placas con microprocesadores comerciales que responden normalmente a un estándar como el PC104 (placas de tamaño concreto que se interconectan entre sí apilándolas unas sobre otras, cada una de ellas con una funcionalidad específica dentro del objetivo global que tenga el SE). Esta última solución acelera el tiempo de diseño pero no optimiza ni el tamaño del sistema ni el número de componentes utilizados ni el coste unitario. En general, un sistema empotrado simple contará con un microprocesador, memoria, unos pocos periféricos de E/S y un programa dedicado a una aplicación concreta almacenado permanentemente en la memoria. [5]

El término embebido o empotrado hace referencia al hecho de que el microcomputador está encerrado o instalado dentro de un sistema mayor y su existencia como microcomputador puede no ser aparente. Un usuario no técnico de un sistema empotrado puede no ser consciente de que está usando un sistema computador. En algunos hogares las personas, que no tienen por qué ser usuarias de un computador personal estándar (PC), utilizan del orden de diez o más sistemas empotrados cada día. [21]

Los microcomputadores empotrados en estos sistemas controlan electrodomésticos tales como: televisores, videos, lavadoras, alarmas, teléfonos inalámbricos, etc. Incluso un PC tiene microcomputadores empotrados en el monitor, impresora, y periféricos en general, adicionales al CPU del propio PC. Un automóvil puede tener hasta un centenar de microprocesadores y microcontroladores que controlan cosas como la ignición, transmisión, dirección asistida, frenos antibloqueo (ABS), control de la tracción, etc.

Los sistemas empotrados se caracterizan normalmente por la necesidad de dispositivos de E/S especiales. Cuando se opta por diseñar el sistema embebido partiendo de una

placa con microcomputador también es necesario comprar o diseñar placas de E/S adicionales para cumplir con los requisitos de la aplicación concreta.

Muchos sistemas empotrados son sistemas de tiempo real. Un sistema de tiempo real debe responder, dentro de un intervalo restringido de tiempo, a eventos externos mediante la ejecución de la tarea asociada con cada evento. Los sistemas de tiempo real se pueden caracterizar como soft o hard. Si un sistema de tiempo real soft no cumple con sus restricciones de tiempo, simplemente se degrada el rendimiento del sistema, pero si el sistema es de tiempo real hard y no cumple con sus restricciones de tiempo, el sistema fallará. Este fallo puede tener posiblemente consecuencias catastróficas. [12]

Un sistema empotrado complejo puede utilizar un sistema operativo como apoyo para la ejecución de sus programas, sobre todo cuando se requiere la ejecución simultánea de los mismos. Cuando se utiliza un sistema operativo lo más probable es que se tenga que tratar de un sistema operativo en tiempo real (RTOS), que es un sistema operativo diseñado y optimizado para manejar fuertes restricciones de tiempo asociadas con eventos en aplicaciones de tiempo real. En una aplicación de tiempo real compleja la utilización de un RTOS multitarea puede simplificar el desarrollo del software. [12]

Figura 1: Sistemas Empotrados.

2.4 Arquitectura de Sistemas Empotrados

Un PC empotrado posee una arquitectura semejante a la de un PC. Brevemente éstos son los elementos básicos:

- **Microprocesador**

  Es el encargado de realizar las operaciones de cálculo principales del sistema. Ejecuta código para realizar una determinada tarea y dirige el funcionamiento de los demás elementos que le rodean.

- **Memoria**

  En ella se encuentra almacenado el código de los programas que el sistema puede ejecutar así como los datos. Su característica principal es que debe tener un acceso de lectura y escritura lo más rápido posible para que el microprocesador no pierda tiempo en tareas que no son meramente de cálculo. Al ser volátil el sistema requiere de un soporte donde se almacenen los datos incluso sin disponer de alimentación o energía.

- **Caché**

  Memoria más rápida que la principal en la que se almacenan los datos y el código accedido últimamente. Dado que el sistema realiza microtareas, muchas veces repetitivas, la caché hace ahorrar tiempo ya que no hará falta ir a memoria principal si el dato o la instrucción ya se encuentra en la caché. Dado su alto precio tiene un tamaño muy inferior (8 – 512 KB) con respecto a la principal (8 – 256 MB). En el interior del chip del microprocesador se encuentra una pequeña caché (L1), pero normalmente se tiene una mayor en otro chip de la placa madre (L2).

- **Disco Duro**

  En él la información no es volátil y además puede conseguir capacidades muy elevadas. A diferencia de la memoria que es de estado sólido éste suele ser magnético. Pero su excesivo tamaño a veces lo hace inviable para PCs empotrados, con lo que se requieren soluciones como discos de estado sólido. Existen en el mercado varias soluciones de esta clase (DiskOnChip, CompactFlash, IDE Flash Drive, etc.) con capacidades suficientes para la mayoría de sistemas empotrados (desde 2 hasta más de 1 GB). El controlador del disco duro de PCs estándar cumple con el estándar IDE y es un chip más de la placa madre.

- **Disquetera**

  Su función es la de un disco duro pero con discos con capacidades mucho más pequeñas y la ventaja de su portabilidad. Siempre se encuentra en un PC estándar pero no así en un PC empotrado.

- **BIOS-ROM**

  BIOS (Basic Input & Output System, sistema básico de entrada y salida) es código que es necesario para inicializar el computador y para poner en comunicación los distintos elementos de la placa madre. La ROM (Read Only Memory, memoria de sólo lectura no volátil) es un chip donde se encuentra el código BIOS.

- **CMOS-RAM**

  Es un chip de memoria de lectura y escritura alimentado con una pila donde se almacena el tipo y ubicación de los dispositivos conectados a la placa madre

(disco duro, puertos de entrada y salida, etc.). Además contiene un reloj en permanente funcionamiento que ofrece al sistema la fecha y la hora.

- **Chip Set**

  Chip que se encarga de controlar las interrupciones dirigidas al microprocesador, el acceso directo a memoria (DMA) y al bus ISA, además de ofrecer temporizadores, etc. Es frecuente encontrar la CMOS-RAM y el reloj de tiempo real en el interior del Chip Set.

- **Entradas al Sistema**

  Pueden existir puertos para ratón, teclado, vídeo en formato digital, comunicaciones serie o paralelo, etc.

- **Salidas del Sistema**

  Puertos de vídeo para monitor o televisión, pantallas de cristal líquido, altavoces, comunicaciones serie o paralelo, etc.

- **Ranuras de expansión para tarjetas de tareas específicas**

  Pueden no venir incorporadas en la placa madre, como pueden ser más puertos de comunicaciones, acceso a red de ordenadores vía LAN (Local Area Network, red de área local) o vía red telefónica: básica, RDSI (Red Digital de Servicios Integrados), ADSL (Asynchronous Digital Subscriber Loop, Lazo Digital Asíncrono del Abonado), etc. Un PC estándar suele tener muchas más ranuras de expansión que un PC empotrado. Las ranuras de expansión están asociadas a distintos tipos de bus: VESA, ISA, PCI, NLX (ISA + PCI), etc.

Hoy en día existen en el mercado fabricantes que integran un microprocesador y los elementos controladores de los dispositivos fundamentales de entrada y salida en un mismo chip, pensando en las necesidades de los sistemas empotrados (bajo coste, pequeño tamaño, entradas y salidas específicas,...). Su capacidad de proceso suele ser inferior a los procesadores de propósito general pero cumplen con su cometido ya que los sistemas donde se ubican no requieren tanta potencia. Los principales fabricantes son ST Microelectronics (familia de chips STPC), National (familia Geode), Motorola (familia ColdFire) e Intel.

En cuanto a los sistemas operativos necesarios para que un sistema basado en microprocesador pueda funcionar y ejecutar programas suelen ser específicos para los sistemas empotrados. Así nos encontramos con sistemas operativos de bajos requisitos de memoria, posibilidad de ejecución de aplicaciones de tiempo real, modulares (inclusión sólo de los elementos necesarios del sistema operativo para el sistema empotrado concreto), etc. Los más conocidos en la actualidad son Windows CE, QNX, VxWorks de WindRiver, y los sistemas basados en Linux.

2.5 Aplicaciones de un PC Empotrado

Los lugares donde se pueden encontrar los sistemas empotrados son numerosos y de varias naturalezas. A continuación se exponen varios ejemplos para ilustrar las posibilidades de los mismos:

En una fábrica, para controlar un proceso de montaje o producción. Una máquina que se encargue de una determinada tarea hoy en día contiene numerosos circuitos electrónicos y eléctricos para el control de motores, hornos, etc. que deben ser gobernados por un procesador, el cual ofrece un interfaz persona – máquina para ser dirigido por un operario e informarle al mismo de la marcha del proceso. [21]

Puntos de servicio o venta (POS, Point Of Service). Las cajas donde se paga la compra en un supermercado son cada vez más completas, integrando teclados numéricos,

lectores de códigos de barras mediante láser, lectores de tarjetas bancarias de banda magnética o chip, pantalla alfanumérica de cristal líquido, etc. El PC empotrado en este caso requiere numerosos conectores de entrada y salida y unas características robustas para la operación continuada.

Puntos de información al ciudadano. En oficinas de turismo, grandes almacenes, bibliotecas, etc. existen equipos con una pantalla táctil donde se puede pulsar sobre la misma y elegir la consulta a realizar, obteniendo una respuesta personalizada en un entorno gráfico amigable.

Decodificadores y set-top boxes para la recepción de televisión. Cada vez existe un mayor número de operadores de televisión que aprovechando las tecnologías vía satélite y de red de cable ofrecen un servicio de televisión de pago diferenciado del convencional. En primer lugar envían la señal en formato digital MPEG-2 con lo que es necesario un procesado para decodificarla y mandarla al televisor. Además viaja cifrada para evitar que la reciban en claro usuarios sin contrato, lo que requiere descifrarla en casa del abonado. También ofrecen un servicio de televisión interactiva o web-TV que necesita de un software específico para mostrar páginas web y con ello un sistema basado en procesador con salida de señal de televisión.

Sistemas radar de aviones. El procesado de la señal recibida o reflejada del sistema radar embarcado en un avión requiere alta potencia de cálculo además de ocupar poco espacio, pesar poco y soportar condiciones extremas de funcionamiento (temperatura, presión atmosférica, vibraciones, etc.).

Equipos de medicina en hospitales y ambulancias UVI – móvil.

Máquinas de revelado automático de fotos.

Cajeros automáticos.

Pasarelas (Gateways) Internet-LAN.

Y un sin fin de posibilidades aún por descubrir o en estado embrionario como son las neveras inteligentes que controlen su suministro vía Internet, PCs de bolsillo, etc.

2.6 Ventajas de un PC Empotrado

Los equipos industriales de medida y control tradicionales están basados en un microprocesador con un sistema operativo propietario o específico para la aplicación correspondiente. Dicha aplicación se programa en ensamblador para el microprocesador dado o en lenguaje C, realizando llamadas a las funciones básicas de ese sistema operativo que en ciertos casos ni siquiera llega a existir. Con los modernos sistemas PC empotrados basados en microprocesadores i486 o i586 se llega a integrar el mundo del PC compatible con las aplicaciones industriales. Ello implica numerosas ventajas: [21]

- Posibilidad de utilización de sistemas operativos potentes que ya realizan numerosas tareas: comunicaciones por redes de datos, soporte gráfico, concurrencia con lanzamiento de threads, etc. Estos sistemas operativos pueden ser los mismos que para PCs compatibles (Linux, Windows, MS-DOS) con fuertes exigencias en hardware o bien ser una versión reducida de los mismos con características orientadas a los PCs empotrados. [21]

- Al utilizar dichos sistemas operativos se pueden encontrar fácilmente herramientas de desarrollo software potentes así como numerosos programadores que las dominan, dada la extensión mundial de las aplicaciones para PCs compatibles. [21]

- Reducción en el precio de los componentes hardware y software debido a la gran cantidad de PCs en el mundo. [21]

Capítulo 3

El PC104

En este capítulo se describe el sistema empotrado conocido como PC104, conociendo su funcionamiento, su arquitectura y los distintos módulos que lo componen.

3.1 Introducción

El PC104 es un estándar de computador empotrado que define el formato de la placa base (*form factor*) y del bus del sistema, el cual se trata del bus ISA adaptado para satisfacer las necesidades de las aplicaciones empotradas e industriales. Su nombre (PC/104) deriva de su arquitectura PC y del conector de 104 pines. Seleccionando la tecnología PC/104, ingenieros y programadores pueden tomar ventaja de sus conocimientos de hardware y software PC-compatible para desarrollar rápidamente sistemas empotrados. [5]

Los sistemas basados en PC104 son utilizados para varias aplicaciones, incluyendo fábricas, laboratorios, plantas de proceso, vehículos y casi cualquier otro lugar donde los dispositivos deban ser controlados por una computadora programable.

Las especificaciones PC104 mecánicas y eléctricas comunes de los módulos hacen que sean intercambiables con productos de cualquiera de los cientos de fabricantes de dispositivos PC/104 que existen actualmente.

3.2 Características

- Las tarjetas PC104 (90 x 96mm) son mucho mas pequeñas que las tarjetas ISA comparable a un disquete de 3.5”.

- Se apilan una sobre otra mediante conectores o pines (socket) lo que elimina la necesidad de los chasis o placas base.

- Se reducen los requerimientos de alimentación y minimiza las conexiones de circuitos.

- Los sistemas PC104 están diseñados para ser más robustos que los sistemas PC.

- Estos sistemas pueden ser programados con las mismas herramientas utilizadas con las PCs: APIs, compiladores, depuradores, herramientas de desarrollo, sistemas operativos y utilidades, lo que reduce la necesidad de conocimientos y el costo de un desarrollo personalizado.

- Independientemente si el host es una tarjeta PC/104, STD Bus, CompactPCI, VME, ISA o PCI, se pueden apilar hasta 4 módulos.

- Si se necesitan múltiples CPUs en un sistema, se puede considerar tener pilas PC/104 separadas que se comunican utilizando el puerto serie o una tarjeta de red. A veces no es necesario para múltiples CPUs tener acceso constante a memoria compartida y puertos de E/S.

- Los módulos PC/104 se encuentran comercialmente disponibles para un amplio rango de funciones, incluyendo: Tarjetas CPU compatibles con PC completas, Entradas/Salidas analógicas y digitales, Video: VGA, LCD, EL, Frame Grabbers, Redes: Ethernet, CAN bus, ARCNET, Controladores: FDD, IDE HDD, SCSI. [5]

3.3 Versiones

Hay tres versiones de la norma:

Fecha	Nombre	Bus	Versión actual (2005)
1992	PC/104	ISA (AT y XT)	2.5
1997	PC/104-*Plus*	ISA y PCI	2.0
2003	PCI-104	PCI	1.0

Tabla 1: Versiones de PC104

El bus de la versión de 1992 del PC/104 usa 104 pines. Estos pines incluyen todas las líneas normales usadas por el bus ISA, además añade líneas de masa para mejorar la integridad de las señales. La sincronización de las señales y los niveles de tensión son idénticos al bus ISA, pero con menos requisitos de corriente.

De forma similar, el bus del PC/104-*Plus* es una versión compacta del bus PCI.

3.3.1 PC104-Plus

PC/104-plus es básicamente la adición del bus PCI (Peripheral Component Interconnect) al estándar PC/104. PCI permite acceso directo a los dispositivos periféricos al CPU el cual puede mejorar en forma considerable la performance del sistema. PC/104-plus ha llegado justo a tiempo para controladores de video, procesadores y otros dispositivos de alto rendimiento, manteniendo la compatibilidad hacia atrás con PC/104.

La especificación PC/104-plus define la adición de PCI a PC/104 incluyendo detalles de los conectores. El nuevo conector tiene 120 pines con espacio de 2mm. El resultado de PC/104-plus es mejor performance para satisfacer las demandas de sistemas empotrados.

3.4 Arquitectura

El PC104 con el cual se ha realizado este estudio, esta compuesto por 4 módulos apilados uno sobre otro, conectados a una fuente de poder quien suministra la cantidad de energía necesaria para el correcto funcionamiento del PC, todo envuelto en una cubierta aislante diseñada para el PC104.

3.4.1 Modulo del CPU

El modulo del CPU corresponde a un modelo **MOPSlcd7** de la empresa alemana Kontron, la cual se encarga de desarrollar sistemas empotrados en todo el mundo. [1]

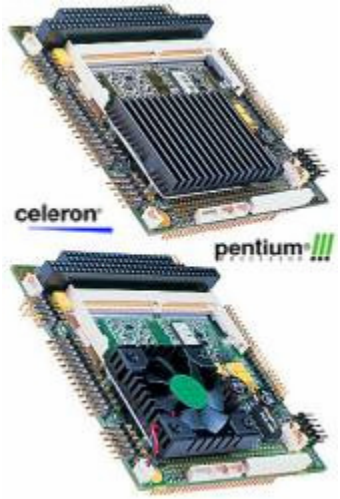

Figura 2: Modulo del CPU

El **MOPSlcd7** corresponde a una sencilla tarjeta que ofrece la oportunidad de obtener un alto rendimiento con un procesador Intel Mobile Pentium III de 700 MHz a un bajo costo. La versión con el procesador Intel es ideal para aplicaciones que requieren un alto rendimiento en el poder de procesamiento. Este modelo es muy versátil, a un bajo costo y aun bajo consumo en los módulos que conforman el estándar del PC104 industrial de 90 mm x 96 mm; el chipset de la tarjeta incluye un VIA Twister-T VT8606 north bridge, un VT82C686B south bridge y un controlador grafico integrado S3 ProSavage4. [1]

Para la visualización de tareas el controlador grafico S3 ProSavage4 abarca hasta 32 Mb, de arquitectura de memoria compartida (SMA) y un JUMPtec inteligente LVDS interfase (JILI) para una adaptación de los paneles LCD, la salida VGA estándar también esta disponible.

El MOPSlcd7 esta compuesto por lo siguiente:

- Un CPU
- Sistema ROM (BIOS)
- Soporta hasta 512Mb de SRAM
- Controladores para acceso directo a memoria (DMA)
- Contadores
- Controladores de Interrupciones
- Controladores para teclado/ratón
- Interfase para altavoz
- Interfase para disco duro IDE
- Puertos Seriales (COM1 y COM2)
- Puerto Paralelo (LPT1)
- Reloj en tiempo real
- Watchdog timer
- Puertos USB
- Ethernet 10/100Base

Datos específicos Funcionales:

- **Procesador:**

  Procesador Mobile Pentium III de 700 MHz con 256 Kb de cache L2 integrada, esta tarjeta esta equipada con un ventilador para el procesador. [1]

- **BIOS Phoenix, con 512Kb Flash BIOS.**

- **Bus:**

  El diseño del MOPSlcd7 sigue el estándar PC104 y ofrece tanto el bus ISA como el bus PCI (El PC104-Plus estándar es compatible hacia abajo con el PC104 clásico) y permite adaptar cualquier tarjeta que cumpla uno de los dos estándares.

  - **PC104 Bus ISA:** el bus de PC104 consiste en dos conectores que usan 104 pines en total.

    El bus conector XT (64 pines); fila A y B.
    El bus conector AT (40 pines, el cual es opcional para buses de 16 bits); fila C y D.

  - **PC104-Plus Bus PCI:** el MOPSlcd7 ofrece el bus para el PC104-Plus en una fila cuádruple en un conector de 2 mm x 2mm. Esto implementa el estándar de señales de 32 bits del bus PCI.

- **Chipset:**

  El VIA Twister-T chipset consiste en:

  - **Controlador VT8660 (north bridge):**
    Alto rendimiento SMA.
    Integrado un VIA Pro133A y un S3 Pro Svage4 en un simple chip.
    FSB de CPU de 133/100 MHz.
    Alto rendimiento en el controlador de 64 bit DRAM.
    La interfase DRAM corre en modo síncrono o seudo-síncrono con el FSB.
    Acceso simultaneo al CPU, AGP y PCI.
    Soporta tipos de memorias estándares como PC133 y PC100.
    Interfase PCI de 32 bits de 3.3 V con salidas de 5 V.
    Soporta un avanzado sistema del manejo de energía.

- **Controlador VT82C686B (south bridge):**
  Controlador para teclado incluido, con soporte para mouse PS/2.
  Integrado reloj de tiempo real (RTC).
  Controlador USB integrado.
  Controlador IDE integrado.
  Controlador Super I/O integrado.
  Interfase para el bus de manejo del sistema (SM).

- **Fuente de Energía:**

  El MOPSlcd7 esta diseñado para usarse como modulo independiente sin una placa madre. Es necesario tener un conector de energía disponible en la tarjeta para suministrar energía de forma directa. El MOPSlcd7 es una tarjeta de 5 V. Algunos periféricos pueden obtener voltaje adicional desde el conector de energía; el voltaje adicional es de +12 V, -5 V, -12 V, y no es generado dentro de la tarjeta del MOPSlcd7. [1]

- **Super I/O:**

  El dispositivo de super I/O esta integrado en el south bridge del VIA chipset, el cual ofrece las siguientes características:

  2 puertos seriales.
  Un puerto paralelo.
  Un controlador para disco floppy.

- **Memoria:**

  El MOPSlcd7 usa solo Pequeños Módulos de Memoria Duales de contorno Íntegros (SODIMMs) de 144 pines. Un socket esta disponible para 3.3 V con un buffer PC-133 o PC-100 (SDRAM) de hasta 512 Mb. [1]

- **Interfase IDE:**

  El MOPSlcd7 posee una interfase EIDE (modo Ultra DMA 33) que puede manejar 2 disco duros. Cuando 2 dispositivos comparten un simple adaptador, ellos están conectados uno como maestro y el otro como esclavo. Si solo se tiene un dispositivo en el sistema, este debe configurarse como maestro.

  La interfase IDE esta disponible a través de un conector J3 (44 pines). Esta interfase esta diseñada en una rejilla de 2 mm, la conectividad de un disco duro de 2.5” es opcional. [1]

- **Interfase USB:**

  La interfase USB del MOPSlcd7 esta integrada en el VT82C686B south bridge, este viene con 2 puertos USB, los cuales sigue las especificaciones UHCI.

- **Ethernet:**

  El MOPSlcd7 usa un controlador Davicom DM9102A. Los controladores de red soportan interfaces 10/100Base. Se puede activar en la tarjeta el soporte para bootear el sistema vía ethernet.

- **Interfase Grafica:**

  El MOPSlcd7 usa un chip integrado S3 ProSavage, un controlador de video de alta velocidad, el cual esta contenido en el VIA Twister chipset (north bridge). El chip S3 ProSavage posee los siguientes beneficios:

  Rendimiento interno AGP 4x
  Optimiza la arquitectura de memoria compartida (SMA)
  Simple ciclo de 128 bit de arquitectura 3D.

Resoluciones 2D/3D hasta 1920x1440.
Soporte para LCD.
Soporta monitores pantalla plana.

- **Interfase de mouse PS/2:**

  El chipset super del MOPSlcd7 soporta un mouse PS/2.

- **Interfase del Teclado:**

  El teclado es una de las características del conector del MOPSlcd7, el cual ofrece 4 funciones. La interfase puede conectar lo siguiente: teclado, altavoz, batería, botón de reset.

- **Interfase del Puerto Paralelo:**

  El MOPSlcd7 incorpora un puerto paralelo que puede ser asignado de forma unidireccional y soporta modos de operación EPP y ECP.

- **Interfase del Floppy:**

  La interfase del floppy del MOPSlcd7 usa un controlador para discos floppy de 2.88 Mb y puede soportar un disco de distintos rangos desde 360 Kb a 2.88 Mb. El controlador es 100% compatible con IBM.

- **Interfase del Ventilador:**

  El conector del fan se usa para colocar un ventilador para enfriar el CPU. El conector y el controlador del sistema soportan la supervisión de la velocidad del ventilador. Este conector solo soporta ventiladores de 5 V.

- **Interfase de Comunicación Serial:**

  2 puertos seriales (COMA y COMB) proveen comunicación serial asíncrona. COMA y COMB soportan modos de operación RS-232. Ellos son compatibles con altas velocidades UART y soportan un buffer de 16 bytes para transferir tasas desde 50 baudios a 115.2 Kbaudios.

- **Especificaciones Mecánicas:**

  - **Conector ISA:** un conector de 2x32 pines y un conector de 2x20 pines.
  - **Conector PCI:** un conector de 4x30 pines.
  - **Dimensiones del Modulo:** 95x90 mm (3.7”x3.5”).
  - **Altura:** 27.2 mm máximo (1.07”).
  - **Peso:** 140 gr.

A continuación se presenta un diagrama de bloques del PC104 modelo MOPSlcd7, donde se pueden observar los distintos componentes mencionados anteriormente:

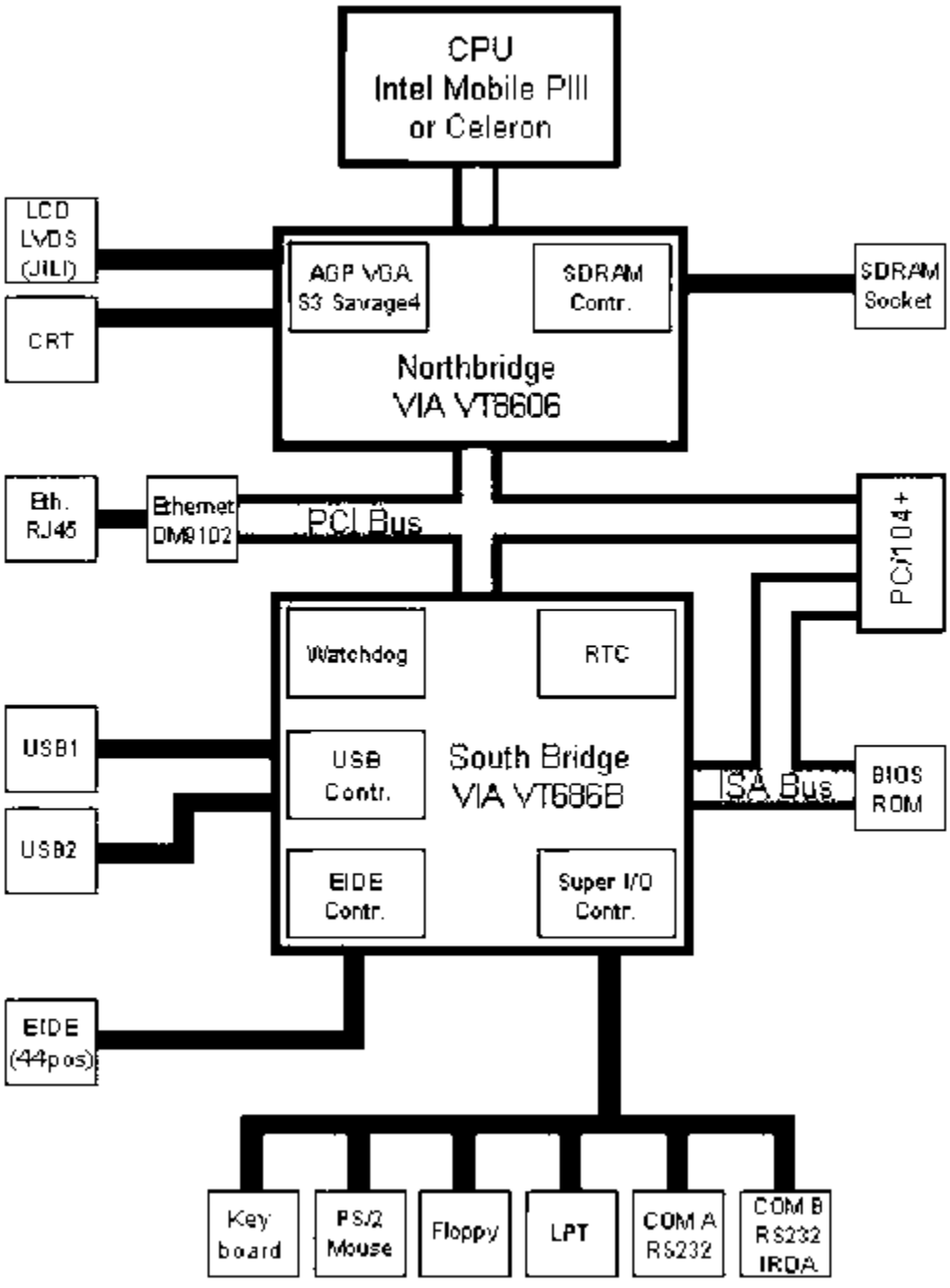

Figura 3: Diagrama de Bloques del MOPSlcd7

3.4.2 Modulo de Disco de Almacenamiento (IDE):

Este modulo se encarga de albergar el disco de almacenamiento masivo (disco duro), en este caso una CompactFlash; el modelo de este modulo es una tarjeta PCM-3116CF de la compañía Canadiense Tri-M Systems and Engineering. [2]

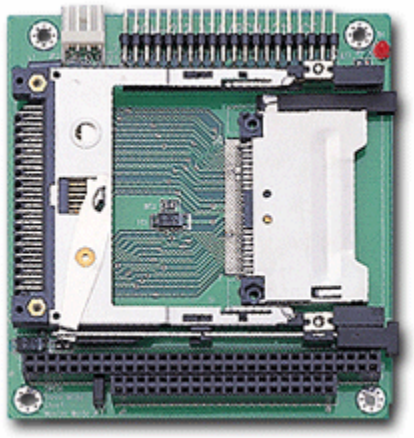

Figura 4: PCM-3116CF

La familia de drivers PCM-3116 brinda las conveniencias de las PCMCIA y las tarjetas CompactFlash para los sistemas computacionales industriales. Todos los modelos de PCM-3116 son compatibles con el estándar de PC104. Las conexiones pueden realizarse de 2 maneras, una a través del conector del PC104 y otra a través de IDE.

La PCM-3116CF es un dispositivo para el PC104 que permite conectar el PC con un controlador IDE a una CompactFlash de lectura escritura. El modulo convierte los 50 pines de una CompactFlash a los 40 pines de la señal IDE. Los BIOS del sistema acceden a la CompactFlash de la misma manera que a cualquier driver IDE. La energía es proporcionada a través de 4 pines del conector de energía de la Compact. La PCM3116CF no requiere de la instalación de un driver adicional y no necesita usar la tarjeta del sistema operativo o un socket de servicio. [2]

En el orden de proporcionar flexibilidad adicional a los sistemas computacionales industriales la PCM-3116CF también provee un conector de PC104. Esto permite que

esta pueda ser conectada a otros módulos de PC104 o conectada a una tarjeta madre con un conector de PC104. Si la PCM-3116CF es conectada usando un conector de PC104, entonces no se necesita conectar cables IDE o conectores de energía, porque la energía y la comunicación de datos son transmitidas a través del conector de PC104. [2]

Especificaciones Más Comunes:

- Cumple con los estándares PCMCIA v. 2.1 / JEIDA 4.1 y CompactFlash.
- Interfase ATA/CompactFlash a IDE.
- Conector IDE de 40 pines.
- Conector de energía estándar de 4 pines.
- Soporta Flash ATA tipo I/II/III, ATA HDD y tarjetas CompactFlash a través de un conector PCMCIA de 68 pines o un conector CompactFlash de 50 pines.
- Consumo de energía de 5V, 70 mA.
- Rango operacional de temperatura 0º C a 70º C.

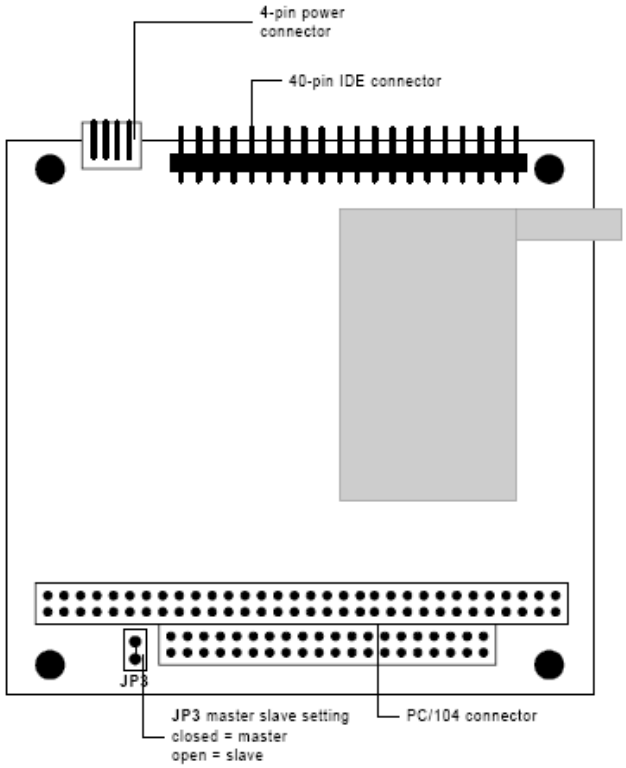

Figura 5: Disposición de conexiones de la PCM-3116CF

3.4.3 Modulo de Red

Este modulo permite conectar el PC104 con otros dispositivos vía ethernet, o permitir el acceso a Internet; este modulo corresponde a un modelo PCM-3660 de la empresa Canadiense Tri-M Systems and Engineering. [3]

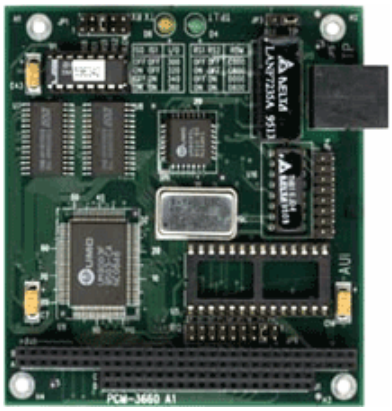

Figura 6: PCM-3660

El PCM-3660 es un modulo de interfase ethernet de 16 bit de alto rendimiento. El modulo automáticamente siente si esta conectado a un sistema PC104 de 8 bit o de 16 bit. El PCM-3660 cumple con los estándares IEEE 802.3 10 Mbps CSMA/CD y es 100% compatible con NE2000. [3]

El modulo incluye un transceiver 10BASE-T y un conector RJ-25; soporta un conector para un transceiver externo para 10BASE-2, 10BASE-5, 10BASEFOIRL, etc.; posee 2 LEDs para indicar el estatus de operación del modulo y de la red.

Hardware:

- Medidas: 90x96 mm (3.6”x3.8”).
- Direcciones de I/O: 300, 320, 340 o 360H.
- Direcciones de Boot ROM: C0000, C8000, D0000 o D800H
- Bus de datos: 8 bit o 16 bit.
- Conectores: conector de 16 bit PC104, conector RJ45 par 10BASE-T.

Soporte de Software:

- Netware 286/386 3.x, 4.x.
- Novell Personal Netware.
- DECnet PathWorks.
- Microsoft LAN manager.
- 3Com 3+Open.
- IBM LAN
- FTP PC/TCP

Información General:

- Energía: +5 V 400 mA max.
- Temperatura: 0º C a 70º C
- Humedad: 10% a 90%

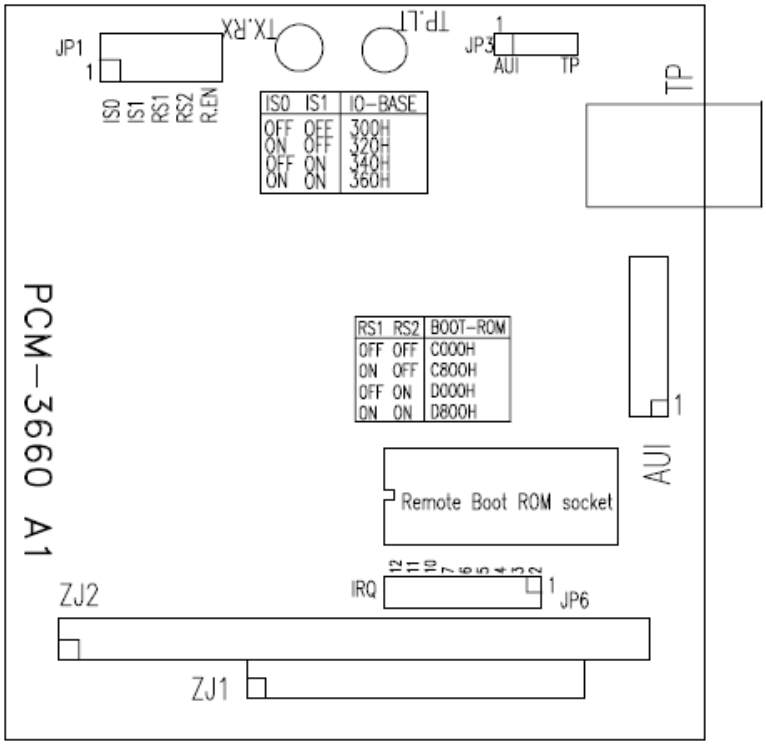

Figura 7. Localización de Componentes en el PCM-3660

3.4.4 Modulo de Comunicación Serial

Este modulo permite la comunicación del PC104 con otros dispositivos vía puerto serial, para cumplir con tareas de adquisición de datos o aplicación de técnicas de control. Este modulo corresponde a un modelo Xtreme/104 desarrollado por la empresa Canadiense Connect Tech Inc. [4]

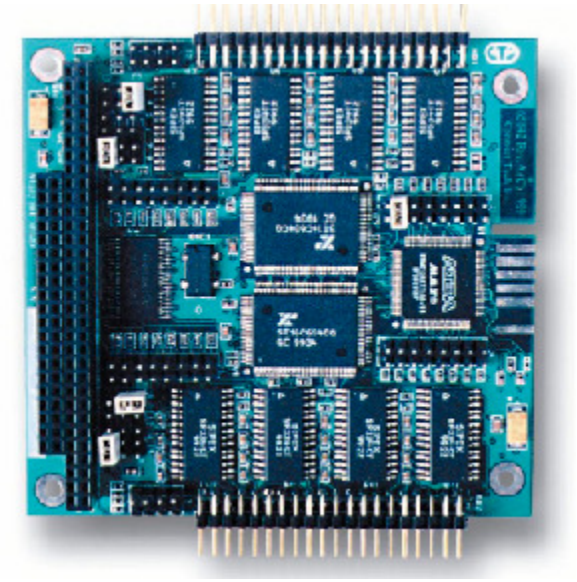

Figura 8: Modulo Xtreme/104

El modulo Xtreme/104 ofrece 4 y 8 puertos seriales asíncronos RS-232 o RS-422/485, para conexión de distintos dispositivos para automatización industrial. El Xtrem/104 es ideal para soluciones de ciertas aplicaciones de control o automatización industrial donde se requiera unos simples o múltiples nodos de comunicación sobre cortas o largas distancias utilizando computadores o dispositivos compatibles con el bus PC104. [4]

Especificaciones:

- Número de Puertos: 4 y 8.
- Interfase Eléctrica: RS-232 y/o RS-422/485, cada puerto tiene un jumper seleccionable.

- Conectores: conector de 40 pines.
- Señales de Control: RS-232: DTR, DSR, RTS, CTS, RI, TxD, RxD, DCD.
- Tasa de Baudios: RS-232: 50 bps a 230.4 Kbps.
- Interrupciones: jumper seleccionable por IRQs
- Alto: 9.60 cm.
- Largo: 10.41 cm.
- Ancho: 1.12 cm
- Requerimientos de Energía: +5 V 100 mA.
- Temperatura: 0º C a 70º C
- Humedad: 95%.
- Bus: bus ISA compatible con el conector de 16 bit de PC104.

3.4.5 Modulo de Alimentación

Este modulo proporciona la alimentación al modulo del CPU del PC104, quien suministra la energía a los demás módulos. Este modulo corresponde a un modelo HE104 de la empresa Canadiense Tri-M Systems and Engineering. [5]

Figura 9: Modulo HE104

Especificaciones:

- Especificaciones Eléctricas: salidas de 5V, -5V, 12V, -12V y entradas entre 6 a 40 VDC.
- Ancho: 3.55”.
- Largo: 3.77”.
- Alto: 0.60”.
- Peso: 172.37 gr.
- Rango de Temperatura: -40º C a 85º C.
- Limpia y Filtrada energía para el bus del PC104.

3.4.6 Armazón del Sistema

Este armazón envuelve el PC104, de tal forma poder aislarlo del contacto con el medio, además de protegerlo de posibles movimientos bruscos. Este armazón corresponde a un modelo VT-6 de la empresa Canadiense Tri-M Systems and Engineering. [6]

Figura 10: Modelo VT-6

Los VT104 corresponden a un armazón de aluminio que pueden resguardar cualquier PC104 o PC104-Plus. La sólida estructura de la pieza proporciona protección ante la vibración. Esta armazón incluye dos placas (frontal y trasera) con variadas configuraciones dependiendo de los distintos dispositivos a conectar al PC104. El VT-6 puede abarcar hasta 8 módulos. [6]

Placa Frontal:

- Modelo: VT-EC29
- Puertos USB: 2
- Puertos DB9: 1
- Puertos PS/2: 2
- Puerto Db25: 1

Figura 11: Placa Frontal.

Placa Trasera:

- Modelo: VT-EC28
- Puertos DB9: 10
- Puertos RJ-45: 2
- Conector de Energía: 1

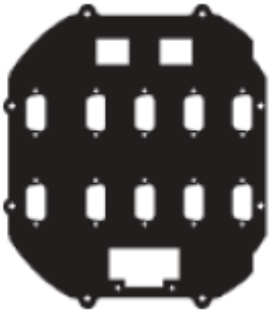

Figura 12: Placa Trasera

Capítulo 4

Sistemas Operativos de Tiempo Real

En este capítulo se muestran algunos de los principales sistemas operativos de tiempo real que se encuentran en el mercado; además se describen algunos de los estándares que estos sistemas deben cumplir.

4.1 Introducción

Los sistemas operativos en tiempo real (RTOS) se encuentran por todas partes son tan útiles como los sistemas operativos familiares OS de Windows, del Mac y Unix, los software de control y los componentes de sistemas que funcionan en PC. Los RTOS trabajan detrás de las escenas de las aplicaciones informáticas y de los componentes de control dentro de los interruptores de la red, de los motores del automóvil, de los paginadores de los teléfonos móviles, de los instrumentos médicos, de la medida industrial y de equipo de control y otros usos.

Una cualidad dominante de un RTOS es la capacidad para aislar usos de modo que si un programa se trunca o actúa de una manera ilegal, el RTOS puede congelar rápidamente el programa, iniciar una recuperación y proteger otros programas o al sistema operativo en sí mismo para y contrarresta las consecuencias de instrucciones errantes. La misma salvaguarda, protege y contrarresta lo que son memorias apiladas los desbordamientos causados por cualquier programa.

Los primeros sistemas operativos de tiempo real (RTOS) estaban diseñados para resolver problemas muy concretos, normalmente en sistemas empotrados tales como automoción, control de vuelo, etc. Actualmente existen muchos sistemas operativos de tiempo real, tanto para sistemas empotrados como no empotrados (RT-Linux, QNX, RTAI, LynxOS, VxWorks, etc.). Y hay una cierta tendencia a estandarizar los RTOS con el objetivo de abaratar los costes y de portar un código de un sistema operativo a otro, un ejemplo de esto podría ser la norma POSIX. [12]

4.2 Características de Los Sistemas de Tiempo Real

Un sistema de tiempo real es aquel en el que la corrección de la computación no solo depende de la corrección de la lógica de la computación, sino también del tiempo en el cuál el resultado es producido. Si los límites temporales del sistema no son alcanzados, se puede decir que una falla del sistema ha ocurrido.

Es decir en un sistema de tiempo no solamente es importante que las tareas se realicen, sino que también deben cumplirse en los plazos de tiempo previstos. Normalmente un STR interactúa con su entorno (objeto o medio que controla), recibe estímulos de este y actúa en consecuencia para producir cambios en el entorno. En estos casos se los denomina sistemas REACTIVOS, ya que reaccionan ante estímulos del medio ambiente. En la figura 13 vemos el esquema básico del funcionamiento de un sistema de tiempo real. [12]

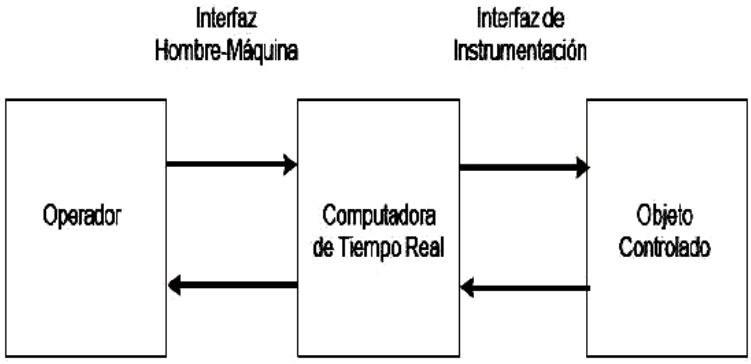

Figura 13: Sistema de Tiempo Real

Si bien el tiempo es el factor determinante para definir el correcto funcionamiento de un STR, erróneamente se tiende a afirmar que cuanto más rápido se realiza una tarea, mejor funciona el STR. Para evaluar el correcto funcionamiento de un STR hay que tener en cuenta los límites según una línea de tiempo, que pueden estar expresados en milisegundos, en algunos casos y en segundos en otros. [12]

Las características fundamentales de los STRs son las siguientes:

- Cierta complejidad
- Fiabilidad y seguridad
- Concurrencia
- Determinismo temporal
- Dispositivos de entrada/salida especiales

4.3 Clasificación de Los Sistemas de Tiempo Real

A continuación se presentan las diferentes clasificaciones en función de características de los sistemas de tiempo real.

- **En base a la importancia que tienen los límites temporales de un STR pueden llegar a clasificarse en tres subgrupos:**

    - Sistemas de tiempo real estricto (hard real-time). Son aquellos en los cuáles no cumplir con alguno de los requerimientos temporales genera una falla general del sistema. Entendiendo como general a una falla completa sin posibilidad de continuar con el funcionamiento. Por lo tanto los límites de tiempo son estrictos y hay veces que es preferible un trabajo imperfecto pero terminado a tiempo. Ejemplo: Control de un reactor nuclear.

    - Sistemas de tiempo real flexible (soft real-time). A diferencia de los anteriores, se pueden llegar a no alcanzar algunos de los límites especificados

pero igualmente el STR continúa su ejecución y la única consecuencia se refleja en una degradación de la calidad de las respuestas. Para estos STR se pueden definir niveles de tolerancia en el momento del diseño de los mismos. Por lo tanto son sistemas en los que los límites de tiempo son flexibles. Ejemplo: Sistema de reserva de pasajes.

- Sistemas de tiempo real firme (firm real-time).Realmente esto no es un subsistemas en si, sino una variación de los sistemas de tiempo real duros, ya que estos sistemas son sistemas de tiempo real duro que pueden tolerar pérdidas, si la probabilidad de ocurrencia de las mismas es baja. Ejemplo: Pérdida de una trama de audio o vídeo.

- **Clasificación según las escalas de tiempo:**

  - Basados en reloj. Se invoca y ejecuta repetidamente a intervalos constantes, a partir de una invocación inicial. Por ejemplo: Tareas periódicas.

  - Basados en eventos. Las acciones se realizan a partir de un evento ocurrido en el proceso que controla. Por ejemplo: Las acciones comienzan a partir de que se lee en un sensor.

  - Interactivos invoca y ejecuta a intervalos irregulares; es decir, sus invocaciones ocurren en instantes arbitrarios. Por ejemplo: Un operario metiendo datos.

- **Según la integración con el sistema físico (o con el medio).**

  - Empotrados (o embebidos) Es un sistema de tiempo real que hace parte de un sistema más complejo. Por ejemplo: sistema de control de inyección de combustible de un automóvil.

- No embebidos

    - Orgánicos. Independientes del hardware en que corren.
    - Débilmente acoplados. Pueden correr en otro hardware reescribiendo ciertos módulos.

- **Clasificación según la forma de procesamiento:**

    - Sistemas centralizados. Son aquellos sistemas en los que un único nodo (mono o multiprocesador) es el encargado de atender a todas las tareas, las tareas se comunican a través de memoria compartida (el gasto de tiempo en comunicación es insignificante)

    - Sistemas distribuidos. Son aquellos sistemas en los que hay varios nodos, unidos a través de una red entre los cuales se reparten la atención de los distintos procesos. Las tareas se comunican a través de la red, no hay memoria compartida (el gasto de tiempo en comunicación es importante).

- **Clasificación según la estrategia de planificación de las tareas.**

    - Sistemas estáticos. En estos sistemas todas las tareas, su naturaleza y características son conocidas de antemano, y en tiempo de diseño se planifica la ejecución de las mismas. El sistema no admite la aparición de una nueva tarea sobre la marcha y tienen bajo costo de ejecución.

    - Sistemas dinámicos (o adaptivos). En estos sistemas puede haber un conjunto de tareas conocido de antemano, pero ante la aparición de una nueva tarea, el sistema analiza si la puede garantizar sin afectar a las tareas que ya maneja, y en ese caso la agrega a la lista de tareas. Aquí hay un mayor costo de ejecución, porque el planificador tiene un trabajo de análisis adicional.

4.4 Estándares Aplicados en Sistemas de Tiempo Real

El principal estándar que se aplica a los sistemas de tiempo real es POSIX. El estándar POSIX (realmente son un grupo de estándares) define una interfaz portable para aplicaciones basadas en el popular sistema operativo UNIX. El principal objetivo de este estándar es la portabilidad de las aplicaciones a nivel de código fuente, mediante la unificación de las diferentes versiones del UNIX. Una parte importante del POSIX aborda las necesidades de las aplicaciones de tiempo real. La portabilidad de estas aplicaciones es hoy en día prácticamente imposible debido a la gran cantidad de sistemas operativos y núcleos de tiempo real existentes. Aunque el UNIX no ha sido hasta ahora un sistema operativo para sistemas de tiempo real, es posible adaptarlo a estos sistemas si se le añaden los servicios necesarios, y se eliminan también aquellas funciones que dificultan implementaciones pequeñas y eficientes. En este apartado se hablará de las extensiones de tiempo real del POSIX y cómo estas extensiones permiten abordar las necesidades de aplicaciones con requerimientos de tiempo real. [12]

Los estándares POSIX se pueden agrupar en tres categorías diferentes:

- **Estándares Base:** Definen interfaces del sistema relacionadas con diferentes aspectos del sistema operativo. El estándar especifica la sintaxis y la semántica de estos servicios del sistema operativo, de modo que los programas de aplicación puedan invocarlos directamente. El estándar no especifica cómo se implementan estos servicios; de este modo, los desarrolladores de sistemas pueden elegir la implementación que crean más conveniente y así competir entre ellos, siempre que cumplan la especificación de la interfaz. Todos los estándares base desarrollados hasta el momento lo han sido para lenguaje C. La Tabla 2 y la Tabla 3 muestran los estándares base que están siendo desarrollados por los grupos de trabajo del POSIX.

POSIX.1	Interfases del sistema (estándar básico)
POSIX.2	/emphShelll y utilidades
POSIX.3	Métodos para medir la conformidad con POSIX
POSIX.4	Extensiones de tiempo real
POSIX.4a	Extensión de /emph threads, o múltiples flujos de control
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 /emphbatch
POSIX.17	Servicios de directorios

Tabla 2: Lista de estándares base del POSIX

POSIX.1	Interfases del sistema (estándar básico)
P1224	Servicios de mensajería electrónica (X.400)
P1224.1	Interfase para portabilidad de aplicaciones X.400
P1238	Interfase de comunicaciones OSI
P1238.1	Interfase OSI de transferencia de ficheros
P1201.1	Interfase gráfica a usuario (ventanas)

Tabla 3: Estándares base POSIX adicionales.

- **Interfaces en diferentes lenguajes de programación:** Son estándares secundarios que traducen a un lenguaje de programación concreto los estándares base. Los lenguajes utilizados hasta el momento son Ada, Fortran 77, y Fortran 90, además del lenguaje C, en el que se han especificado hasta el momento los estándares base. La Tabla 4 muestra las interfases POSIX que están actualmente en desarrollo para diferentes lenguajes de programación.

- **Entorno de Sistemas Abiertos.** Estos estándares incluyen una guía al entorno POSIX y los perfiles de entornos de aplicación. Un perfil de aplicación es una lista de los estándares POSIX, con especificación de las opciones y parámetros necesarios, que se requieren para un cierto entorno de aplicación. El objetivo principal de los perfiles de aplicación es conseguir un conjunto pequeño de clases de implementaciones de sistemas operativos bien definidas y que sean apropiadas para entornos particulares de aplicaciones. La Tabla 5 muestra la lista de estándares que están siendo desarrollados en este grupo.

POSIX.5	Interfases Ada
POSIX.9	Interfases Fortran 77
POSIX.19	Interfases Fortran 90
POSIX.20	Interfases Ada para las extensiones de tiempo real

Tabla 4: Lista de interfaces POSIX para diferentes lenguajes de programación

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

Tabla 5: Lista de estándares POSIX de entornos de aplicaciones

La necesidad del desarrollo de un estándar de sistema operativo se deriva del hecho de que, aunque el UNIX es un estándar de facto, hay suficientes diferencias entre las diferentes implementaciones para que las aplicaciones no sean completamente portables. Más aún, si una aplicación UNIX puede necesitar ciertos cambios para ser portada a una plataforma diferente, la portabilidad de las aplicaciones de tiempo real es muchísimo

más difícil, ya que existe una gran diversidad de sistemas operativos de tiempo real. El UNIX no es un sistema operativo de tiempo real, y no existe un estándar de facto para estas aplicaciones.

Una vez enumerados todos los estándares POSIX, vamos a centrarnos en los de tiempo real y se comentaran las principales características que engloban cada uno de ellos.

POSIX.4: Extensiones de tiempo real. Define interfaces para soportar la portabilidad de aplicaciones con requerimientos de tiempo real. En este estándar se engloban lo siguiente: 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 Entrada/Salida Asíncrona Entrada/ Salida Sincronizada.

POSIX.4a: Extensión de threads. Define interfaces para soportar múltiples threads o flujos de control dentro de cada proceso POSIX. Aquí se engloba lo siguiente: Control de Threads Planificación de Threads Sincronización de Threads Otras Funciones (se definen otras funciones para el manejo de datos asociados a cada thread, la cancelación de threads, envío de señales a threads, así como versiones reentrantes de otras funciones definidas en el POSIX.1.).

POSIX.4b: Extensiones adicionales de tiempo real. Define interfases para soportar servicios de tiempo real adicionales. Tiempos Límite (TimeOuts) Relojes de Tiempo de Ejecución Servidor Esporádico Control de Interrupciones Control de Dispositivos de Entrada/Salida Creación de Procesos.

POSIX.5: Interfaz para lenguaje Ada. Define las interfaces básicas del sistema para lenguaje Ada.

POSIX.20: Interfaz Ada para las extensiones de tiempo real.

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

Los estándares base POSIX.4, POSIX.4a y POSIX.4b están especificados para lenguaje C. Existe un grupo de trabajo en el POSIX dedicado a la especificación de interfaces Ada que ha creado el estándar POSIX.20, que es una extensión de tiempo real pero especifica para Ada.

4.5 Generalidades de Los Sistemas Operativos de Tiempo Real

Un sistema operativo para tiempo real debe ser un sistema operativo capaz de garantizar los requisitos temporales de los procesos que controla.

Los sistemas operativos convencionales no son apropiados para controlar sistemas de tiempo real, debido a que su comportamiento no es determinista y a que no son capaces de garantizar los tiempos de respuestas.

Los S.O.T.R. deben presentar los siguientes requisitos:

Determinismo

Un SO es determinista si realiza las operaciones en instantes fijos y predeterminados o en intervalos de tiempo predeterminados. En un S.O.T.R. las solicitudes de servicios vienen dictadas por eventos y temporizaciones externas, por lo tanto el que pueda satisfacer las peticiones de forma determinista depende de:

- La velocidad de respuesta a las interrupciones.
- Si el sistema posee suficiente capacidad de gestionar todas las peticiones en el tiempo requerido.

Entonces, se considerará determinista si se puede calcular el retardo máximo que se produce desde la llegada de la interrupción de un dispositivo hasta que se comienza el servicio, es decir, si se puede calcular el máximo tiempo de una llamada del sistema.
En los S.O.T.R. esta cota máxima es del orden de microsegundos, mientras que en los SO pueden estar en el rango de milisegundos. [12]

Sensibilidad

El determinismo es el tiempo que tarda el sistema en reconocer la interrupción, mientras que la sensibilidad es el tiempo que consume el S.O.T.R. en dar servicio a la interrupción. Incluye lo siguiente:

- Tiempo necesario para iniciar la gestión de la interrupción y comenzar la ejecución de su rutina de tratamiento (ISR).
- Tiempo para ejecutar la ISR.
- El efecto del anidamiento de interrupciones.

Control del usuario

El control de usuario es mayor en los S.O.T.R. ya que el usuario tiene control sobre las prioridades de las tareas de la aplicación, puede especificar aspectos de paginación o intercambio de procesos y en sistemas distribuidos también puede controlar la asignación de procesos a procesadores.

Fiabilidad

Un fallo de un sistema normal se soluciona arrancando de nuevo el sistema, pero los fallos en un S.O.T.R. pueden dar lugar a resultados catastróficos

Tolerancia a fallos

Un STR debe diseñarse para responder incluso ante fallos. La tolerancia a fallos es una característica que hace referencia a la capacidad de un sistema de conservar el máximo rendimiento y los máximos datos posibles en caso de fallo.

Otro aspecto importante en la estabilidad, esto es que si no se cumplen los plazos de algunas tareas, el S.O.T.R. debe garantizar que al menos se cumplen los plazos de las más críticas.

Por lo tanto para que un S.O.T.R. pueda cumplir todos los requisitos anteriormente explicados, el S.O.T.R. debe poder ofrecer las siguientes facilidades:

- **Concurrencia:** Procesos ligeros (threads) con memoria compartida.
- **Temporización:** Medida de tiempos y ejecución periódica.
- **Planificación**: Es el elemento fundamental del S.O.T.R., hay varios tipos: Prioridades fijas con desalojo o acceso a recursos con protocolos de herencia de prioridad.
- **Dispositivos de E/S:** Acceso a recursos hardware e interrupciones.

Los SO puede ofrecer tres tipos de estructuras internas, cada una tiene sus ventajas y sus inconvenientes por lo que se describen brevemente:

- **Monolíticos:** El S.O está compuesto de una única pieza de código, quizás dividida en módulos.

    - Ventajas

        - Incluye toda la funcionalidad necesaria en el núcleo.
        - De fácil instalación y configuración.

- Inconvenientes

    - Difícil de reducir para el caso de sistemas empotrados.
    - Difícil de depurar. Las correcciones de errores del S.O. pueden producir nuevos errores.
    - Cambios en el S.O. Afectan normalmente a muchas partes del mismo.

- **En Capas:** El S.O. está dividido en capas superpuestas, con distintos niveles de abstracción.

    - Ventajas

        - Más configurable y aislado que los S.O. monolíticos.

    - Inconvenientes

        - Aún difícil de reducir y de depurar.
        - No todos los servicios del S.O. se prestan a estar divididos en capas.

- **Cliente/Servidor:** El S.O. está compuesto de diversos servidores, módulos aislados que proporcionan servicios a otros módulos llamados clientes. Normalmente existe una pequeña porción del S.O. que incluye la funcionalidad básica (multitarea, interrupciones, etc.), llamada micronúcleo.

    - Ventajas

        - Muy fácil de depurar.
        - Muy fácil de escalar a las necesidades de la aplicación (muy usados en sistemas empotrados).
        - Permite configurarlo dinámicamente, cargando y descargando servidores.

- Inconvenientes

    - Al estar más fragmentado, se incrementa el coste de las comunicaciones entre partes de la aplicación si éstas pasan por el S.O.
    - Mayor complejidad de configuración (elección de los servidores adecuados).

4.6 Características de Rendimiento

El criterio fundamental de evaluación del rendimiento de un sistema operativo de tiempo real es la latencia y el periodo del jitter ante un evento. Un evento es cualquier tipo de interrupción, tanto interna, como externa.

Latencia en un evento. Un evento puede ser tanto una interrupción hardware como una interrupción software. La latencia ante una interrupción hardware es el tiempo desde que se produce la interrupción hasta que se ejecuta la primera instrucción de la rutina de tratamiento. Puede haber retrasos debido al acceso al bus. La latencia ante una interrupción software es el tiempo desde que la señal es generada hasta que la primera instrucción de la tarea es ejecutada. Aquí el valor depende únicamente del acceso a los registros del procesador. En la figura 14 podemos ver la latencia de un evento. [12]

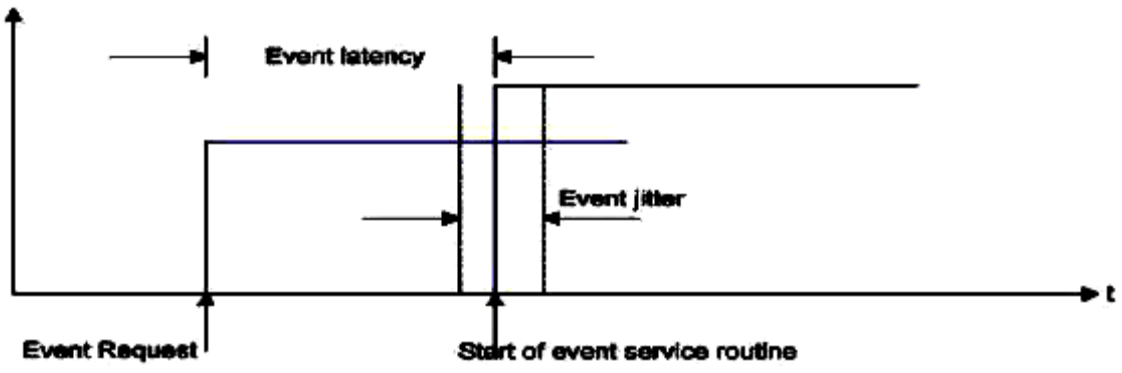

Figura 14. Latencia de Un Evento

Periodo del Jitter. El periodo del jitter se refiere a las variaciones en el tiempo que experimenta una tarea cuando se ejecuta de manera repetitiva. En la figura 15 se puede ver este periodo. [12]

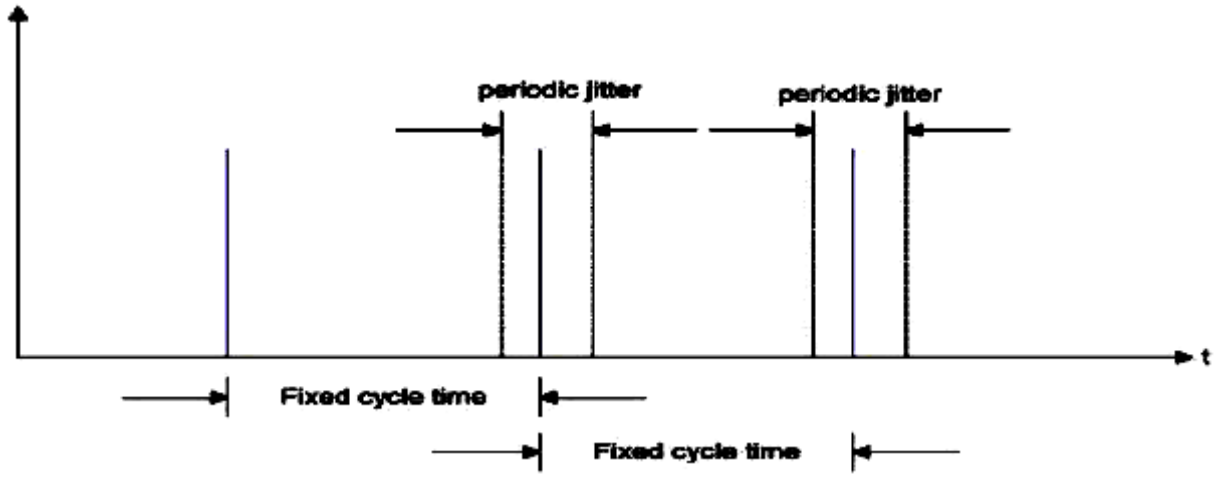

Figura 15: Periodo de Jitter de Un Evento

4.7 Arquitectura de Los Sistemas Operativos de Tiempo Real

El objetivo de un sistema operativo de tiempo real es reducir la latencia y el jitter en las interrupciones, tanto internas como externas, al orden de microsegundos. Es decir, la parte fundamental para convertir un sistema operativo de propósito general en un sistema operativo de tiempo real es el manejo de las interrupciones. El procesamiento de interrupciones en el kernel estándar esta divido en 2 tareas, una tarea que se encarga de leer los datos del dispositivo físico y escribirlos en un buffer, es lo que se conoce como manejador de interrupciones, y otra tarea que se encarga de pasar los datos del buffer a otro para que sean accesible por el kernel. [12]

Con este esquema, cuando el manejador esta ejecutando, todas las interrupciones están inhibidas con el siguiente retardo impredecible en el servicio de otras interrupciones que se puedan haber producido y por tanto en los valores de latencia y jitter.

Para conseguir reducir la latencia y el jitter se han desarrollado distintas alternativas que modifican el kernel de Linux en este aspecto fundamentalmente. Actualmente hay dos corrientes de diseño para los sistemas operativos de tiempo real:

4.7.1 Atención prioritaria en el kernel estándar

Esta metodología modifica el kernel en profundidad de forma que los procesos de kernel se ejecutan con máxima prioridad de forma que puedan interrumpir a procesos de menor prioridad en el acceso a los recursos que necesiten. Esta metodología implica cambios en los manejadores de interrupciones para que las interrupciones de alta prioridad no sean bloqueadas por el manejador de interrupciones mientras esta manejando otra de menor prioridad. El resultado de esta metodología es una latencia y un jitter del orden de 1 milisegundo en un Pentium a 100 Mhz.

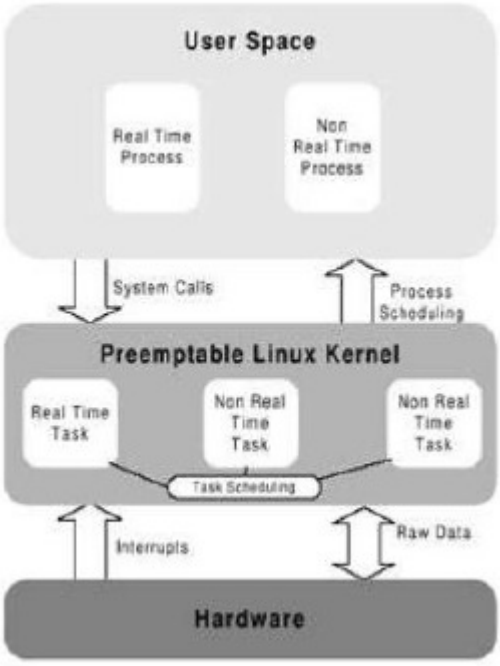

Figura 16. Arquitectura con kernel Apropiativo

Como se puede observar en la figura 16, la tarea de tiempo real esta controlada por el planificador del kernel y es una más de las tareas que controla el kernel. Esta tarea hace referencia a los procesos de tiempo real en el espacio de usuario.

El planificador sabe que las tareas de tiempo real tienen mayor prioridad que las tareas que no son de tiempo real. Como se puede comprobar esta metodología es adecuada para aplicaciones de audio y vídeo donde el periodo de interrupciones es del orden de 1 milisegundo, pero inadecuado cuando se habla de menos de 1 milisegundo. Esta es una de las limitaciones de esta estrategia.

4.7.2 Modificaciones sobre el kernel estándar

Existen 4 estrategias de modificación del kernel de Linux para proveer capacidades de tiempo real. Tres de ellas implican añadir un segundo kernel (kernel dual) para manejar las tareas de tiempo real y el cuarto implica modificar directamente el código del kernel para añadir características de tiempo real.

4.7.2.1 Microkernel

Esta estrategia añade un segundo kernel que en realidad es una capa interfaz entre el hardware y el kernel estándar, lo que se llama tradicionalmente HAL "Hardware Abstraction Layer".

Esta capa, micro kernel, controla la ejecución de las tareas de tiempo real y ejecuta el kernel estándar como una tarea en background, es decir, el kernel estándar sólo ejecuta cuando no hay tareas de tiempo real pendientes.

Por lo tanto el microkernel intercepta las interrupciones hardware y asegura que las tareas de tiempo real ejecuten con la mayor prioridad posible de forma que la latencia se minimice, para conseguir que se cumplan los Ejemplo de implementación de esta metodología son RTLinux y RTAI.

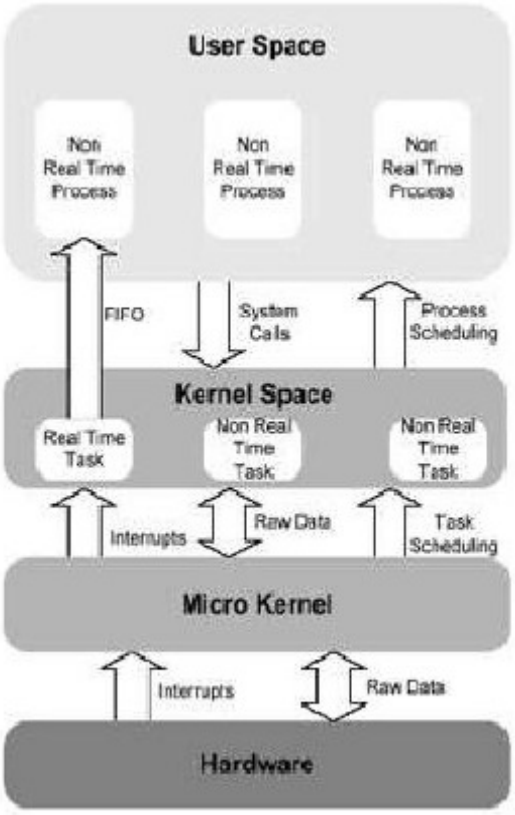

Figura 17: Arquitectura con Micro-Kernel

4.7.2.2 Nanokernel

Esta estrategia es similar a la primera, microkernel, pero consigue evitar la patente que tiene Yodaiken sobre ésta, de forma que este nanokernel únicamente captura las interrupciones hardware y permite la ejecución paralela de varios sistemas operativos por encima de él. En este sentido, esta estrategia no desemboca directamente en un sistema operativo de tiempo real, sino que simplemente es una capa intermedia entre el hardware y un sistema operativo, que puede ser de tiempo real ó no. En la figura 18 podemos ver un esquema de esta arquitectura.

Una implementación de esta estrategia es ADEOS. Los desarrolladores de este proyecto fueron precisamente los que pusieron el nombre de nanokernel para diferenciarlo claramente de la estrategia de micro-kernel patentado por Yodaiken.

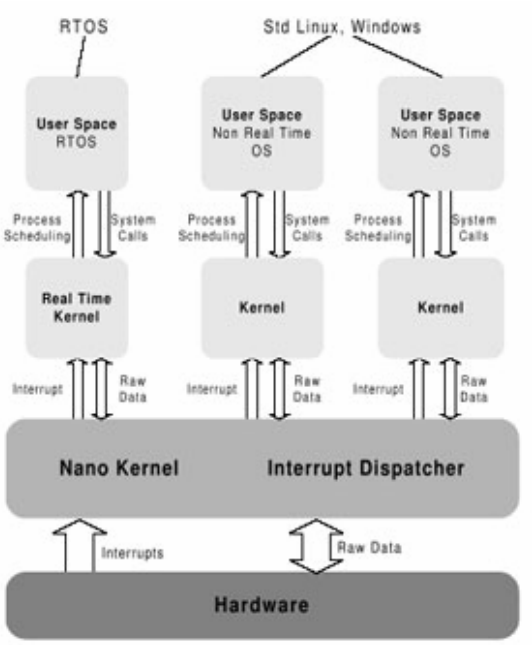

Figura 18: Arquitectura con NanoKernel

4.7.2.3 Extensión con un nuevo kernel de acceso a los recursos

Esta estrategia añade un kernel de forma que éste proporciona una puerta de acceso a los recursos, como pude ser el sistema de ficheros, el puerto paralelo, etc. tanto para el kernel estándar como para los procesos de usuario. El recurso kernel no sólo captura las interrupciones sino que proporciona un mecanismo donde los programas de usuario pueden requerir, reservar y garantizarse un porcentaje finito de los recursos como pueden ser de CPU, memoria, etc.

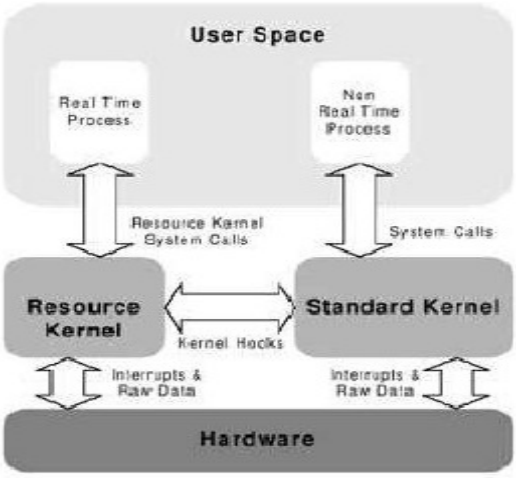

Figura 19: Arquitectura con Recurso-Kernel

4.7.2.4 Extensiones POSIX de tiempo real añadidas al kernel

Esta estrategia consiste en modificar directamente el kernel estándar de Linux para añadir librerías que implementan las extensiones de tiempo real de POSIX. El resultado es un kernel conforme al estándar IEEE 1003.1d. No se añade un segundo kernel, las modificaciones realizadas al kernel consisten en la implementación de relojes, señales, semáforos, memoria compartida, planificador por prioridades, etc. según lo especificado en IEEE 1003.1d.

4.8 Estudio de Los Principales Sistemas Operativos de Tiempo Real

Se va a realizar una breve descripción de la arquitectura, licencia y entorno para el cual están diseñados distintos S.O.T.R. Algunos de éstos están en continuo desarrollo y otros por el contrario están un poco estancadas ó definitivamente olvidados, pero que son nombrados porque nacieron con el objetivo de proporcionar una nueva característica ó una nueva filosofía de diseño que otros S.O.T.R. no proporcionaban y que otras han seguido posteriormente.

En la tabla 6 se proporciona información acerca del tipo de licencia bajo la que se distribuye (GPL, Comercial,…), su estado actual (desarrollo, parada,...) y su posible ultima versión del kernel en la que se basa.

Como se puede observar en la tabla 6, en el mercado hay tanto distribuciones comerciales y de código abierto, como distribuciones libres. También se observa, que actualmente, las distribuciones libres ADEOS y RTAI son las que están más actualizadas porque poseen versiones validas para la versión 2.6 del kernel estándar de Linux. El resto ó están muy dejadas ó tardan bastante en sacar nuevas versiones.

Distribución	**Licencia**	**Estado**	**Arquitectura**	**Última Versión de *kernel***
ADEOS	GNU/GPL	Activo	*Nano-kernel*	2.4.24 - –>2.6.x
RTAI	GNU/GPL	Activo	*Micro-kernel*	2.4.xx - –>2.6.x
RTLinux	GNU/GPL && *Open RTLinux Patent*	Parado	*Micro-kernel*	2.4.21
	Comercial	Desconocido	*Micro-kernel*	Desconocido
ART-Linux	Privado	Desconocido	*Micro-kernel*	Desconocido
KURT	Desconocida. *Open Source*	Activo	Extensiones POSIX	2.4.18
Linux/RK	Desconocida. *Open Source*	Activo	*Recurso-kernel*	2.4.18
Qlinux	GNU/GPL	Activo	QoS	2.4.4
RED-Linux	Desconocida	Desconocido	Extensiones POSIX	2.0.35
BlueCat-RT	Comercial. *Open Source*	Activo	*Micro-kernel* + *Preemptable (kernel estándar)*	Desconocido
RedHawk	Comercial	Desconocido	*Preemptable-kernel*	Desconocido
REDICE-Linux	Libre de *Royaltis*. Soporte comercial	Activo	*Micro-kernel* + QoS (*kernel* estándar)	2.4.18
TimeSys Linux	Comercial. Parte es GPL	Activo	*Preemptable-kernel* + Extensiones POSIX	2.4.21
Linux-SRT	GNU/GPL	Abandonada	Extensiones POSIX	2.2.15
QNX	Comercial	Activo	*Micro-kernel*	Desconocido
MaRTE	GNU	Activo	Extensiones POSIX	Desconocido
VxWorks	Comercial	Activo	*Micro-kernel*	Desconocido
Chimera	GNU	Activo	*Preemptable-kernel*	Desconocido

Tabla 6: Lista de Sistemas Operativos de Tiempo Real

4.8.1 RTLinux

RTLinux es un sistema operativo de tiempo real que ejecuta Linux como un thread de menos prioridad que las tareas de tiempo real. Con este diseño, las tareas de tiempo real

y los manejadores de interrupciones nunca se ven retrasados por operaciones que no son de tiempo real.

La primera versión de RTLinux estaba diseñada para ejecutarse en la plataforma x86 y proporcionaba una pequeña API y un pequeño entorno de programación. La versión 2, que fue totalmente rescrita, fue diseñada para el soporte de multiprocesamiento simétrico (SMP) y para ser ejecutada en una amplia variedad de arquitecturas.

RTLinux proporciona la capacidad de ejecutar tareas de tiempo real y manejadores de interrupciones en la misma máquina que el Linux estándar. Estas tareas y los manejadores ejecutan cuando se necesitan en detrimento de lo que estuviera ejecutando Linux. El peor caso de tiempo es entre que se detecta la interrupción hardware y el procesador ejecuta la primera instrucción del manejador de la interrupción. Este tiempo es del orden de los 10 microsegundos en la plataforma x86. [12]

Arquitectura

RTLinux es un pequeño y rápido sistema operativo que sigue el estándar POSIX 1003.13: sistema operativo de tiempo real mínimo.

RTLinux añade una capa de abstracción hardware entre el kernel estándar de Linux y el hardware de la máquina. Esta es la arquitectura de micro-kernel. En la figura 11 podemos ver la arquitectura de RTLinux detallada en capas.

Hay 3 modificaciones principales en el kernel de Linux con el objetivo de que RTLinux tenga el control del hardware de la máquina:

- Control directo de las interrupciones hardware.
- Control del reloj hardware e implementación de un reloj virtual para Linux.
- El control de las interrupciones por parte de Linux es reemplazado por 2 funciones que permiten activar ó desactivar las interrupciones, pero las virtuales.

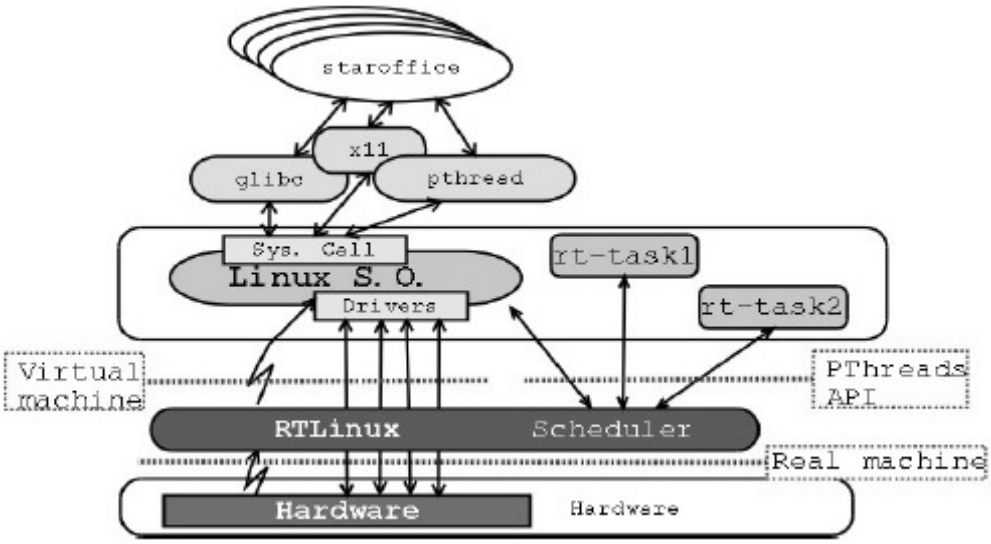

Figura 20: Arquitectura de RTLinux.

Características

- Soporte de múltiples arquitecturas y válida para arquitecturas multiprocesador.
- Gestión de procesos: Planificación, soporte de threads periódicos, amplio rango de prioridades, creación y borrado de threads, etc.
- Gestión de memoria: No protección de memoria en el kernel y no asignación de memoria de forma dinámica.
- Comunicación entre procesos: Semáforos, Mutex, control de inversión de prioridades, memoria compartida y FIFO's.
- Tiempo y relojes: Resolución de nanosegundos. No relojes de usuario. Facilidades para añadir nuevos relojes hardware.
- Programación de drivers: Se proporcionan funciones de acceso a dispositivos.
- No proporciona herramientas de calidad de servicio.

4.8.2 ART Linux

ART Linux es una extensión de tiempo real sobre Linux basado en RTLinux y desarrollado por Youichi Ishiwata.

ART Linux es un parche sobre el kernel estándar de Linux pero que actualmente no se distribuye bajo la licencia GPL ya que contiene código desarrollado por Ishiwata en exclusiva. El que no se distribuya bajo la licencia GPL es debido a que Ishiwata trabaja para el gobierno y existe una ley de propiedad intelectual del gobierno.

La arquitectura es similar a la de RTLinux por tanto tienen arquitectura de kernel dual.

Ventajas de ART Linux:

- ART Linux permite reutilizar los drivers de dispositivos existentes en el Linux estándar así como las aplicaciones.
- Compatibilidad a nivel de fuente de los drivers con el Linux estándar. Es decir, si recompilas el driver, puede seguir siendo usado por las tareas de tiempo real.
- No hay inversión de prioridades. Se resuelven automáticamente los problemas de inversión de prioridades.
- Compatibilidad a nivel binario de los programas de usuario con el Linux estándar. No es necesario recompilar los programas con el objetivo de que sean usados por las tareas de tiempo real.
- Evita indeterminismo en la ejecución de tiempo real debido a las interrupciones ya que trata las interrupciones con una estrategia periódica.
- Las tareas de tiempo real ejecutan en modo usuario pero privilegiado, de forma que se beneficia de la protección de memoria y por tanto las tareas de tiempo real son seguras y no pueden provocar fallos en el kernel.

4.8.3 KURT

KURT junto con UTIME es una extensión al kernel estándar de Linux para proporcionar, bajo demanda, resolución de microsegundos en los relojes y planificación de tiempo real.

Arquitectura

KURT proporciona un manejador de eventos con la facilidad de un planificador de tiempo real con resolución de microsegundos gracias a UTIME.

El kernel estándar de Linux proporciona 3 políticas de planificación diferentes, que son: SHED_FIFO, SHED_RR y SHED_OTHER, que son suficientes para la mayoría de aplicaciones que no requieren una granularidad de resolución muy fina. Un valor alto de prioridad no garantiza a un proceso su planificación, por eso KURT proporciona un mecanismo de planificación bajo demanda para garantizar a los proceso el acceso a los recursos que necesite y en un orden explícito.

Todas las interacciones con el subsistema KURT se realizan a través de un psudodispositivo (/dev/kurt), que únicamente proporciona 3 operaciones: open, close y ioctl. Cada vez que se quiera usar KURT simplemente hay que abrir una instancia del dispositivo y cuando se termina, simplemente se cierra. Para facilitar la programación de aplicaciones KURT dispone de un API que proporciona las siguientes 3 categorías de rutinas:

- Operaciones generales y de utilidad. Un ejemplo de utilidad es la función kurt_open(), que abre una instancia del psudodispositivo.
- Inicialización, registro y control de procesos. Un ejemplo es la función rt_suspend(), que suspende un proceso por un determinado plazo de tiempo. Antes de registrar los procesos de tiempo real se debe determinar que tipo de planificación van a soportar, puesto que KURT soporta un amplio abanico de modos y niveles de planificación y combinación entre ellos.
- Planificación. Un ejemplo es la función set_sheduling_task(), que registra los procesos actuales en el planificador de tareas de KURT.

4.8.4 Linux/RK (Linux/Resource Kernel)

Linux/RK significa Linux/Resource Kernel y consiste en incorporar extensiones de tiempo real a Linux mediante una abstracción llamada recurso kernel.

Arquitectura.

- Las aplicaciones que requieran acceder a alguno de los recursos se comunican con el recurso-kernel y realiza una reserva. Si es aceptada obtendrá el recurso cuando lo haya solicitado.
- Un recurso-kernel es un kernel de tiempo real que proporciona el oportuno y garantizado acceso a los recursos del sistema para las aplicaciones que lo requieran.

Características.

- Reserva de ancho de banda en acceso a los discos.
- Reserva de ancho de banda de la red.
- Co-planificación de varios recursos.
- Integración con Java para tiempo real.
- Lista de control de recursos.

4.8.5 Qlinux

Qlinux es un kernel de Linux que proporciona calidad de servicio garantizada para requerimientos de tiempo real flexible. Por tanto no es un sistema operativo específico de tiempo real, sino que esta orientado hacia aplicaciones multimedia que requieren una determinada calidad de servicio.

Arquitectura.

Usa una arquitectura precursora a la de "kernel preemptable", pero en este caso no solo aplicada al acceso de la CPU sino que también aplicada al acceso a la red y al disco.

Características.

- H-SFQ (Hierarchical Start Time Fair Queuing) para el planificador de la CPU. El planificador activa una planificación jerárquica de forma que asigna ancho de banda de la CPU de forma justa entre las aplicaciones o clases de aplicaciones.
- H-SFQ para el planificador de paquetes de red. De manera similar a antes, pero en este caso, proporciona transferencias garantizadas y una asignación de ancho de banda para los paquetes de flujos individuales o clases de flujos.
- Algoritmo de planificación del disco (Cell disk scheduler). Soporta múltiples clases de aplicaciones tales como interactivas, best-effort, transferencias intensivas, tiempo real flexible, etc.

4.8.6 RED-Linux

RED-Linux es una versión de Linux para tiempo real y empotrado. Proporciona capacidades adicionales al kernel estándar de Linux para proporcionar comportamiento de tiempo real.

Fundamentos.

- Un núcleo pequeño.
- Un tiempo de respuesta rápido para las tareas.
- Modularidad y Planificador de CPU reemplazable en tiempo de ejecución.
- Un marco general de planificación o General Sheduling Framework (GSF).

4.8.7 BlueCat RT

BlueCat RT es una versión de Linux para tiempo real y empotrado. Es una solución híbrida entre el sistema operativo embebido BlueCat (LinuxWorks) y el sistema RTLinux/Pro (FSMLabs).

Características.

Estabilidad. Es una distribución estable porque se basa en un sistema operativo embebido con cierto grado de comercialidad y por tanto se diseña para que sea flexible, escalable y con productividad inmediata.

Flexibilidad. Soporte de un completo rango de configuraciones, desde pequeños dispositivos para el consumidor hasta grandes sistemas multiprocesadores.

4.8.8 RedHawk

Es una versión de tiempo real del código abierto de Linux desarrollada por la empresa Concurrent Computer Corporation's RedHawk bajo los estándares industriales y POSIX. RedHawk es compatible con la distribución Red Hat Linux proporcionando alto rendimiento de entrada/salida, tiempo de respuesta garantizado a evento externos y comunicación entre procesos optimizada.

Arquitectura.

El sistema de administración, las utilidades y los comandos del nivel de usuario son los estándares proporcionados por Red Hat, en cambio, para lograr rendimiento de tiempo real, reemplaza el kernel de Red Hat por uno que es multithread, con un kernel completamente apropiativo, es decir capaz de interrumpir a tareas con menor prioridad, y de baja latencia.

RedKawk proporciona soporte al multiprocesamiento simétrico, que incluye un control de carga y protección de CPU para maximizar el determinismo y el rendimiento de tiempo real en soluciones críticas.

Características.

- Ambiente completo de desarrollo. Ofrece un conjunto completo de herramientas de desarrollo, desde compiladores a debuggers y herramientas que muestran la traza de ejecución.
- SMP escalable y protección de procesador.
- Multithread y kernel prioritario.
- Planificador basado en la frecuencia.
- Reloj de tiempo real y módulo de interrupciones.
- Herramientas de desarrollo de aplicaciones de tiempo real.
- Depurador de código fuente.
- Analizador de eventos mediante traza de código.
- Simulador de un planificador.
- Monitor de datos de prueba.

4.8.9 REDICE-Linux

REDICE-Linux es un kernel de tiempo real que combina la planificación de tiempo real estricto y un kernel prioritario con la baja latencia de un completo kernel de tiempo real estricto como es el de RTAI. Es decir, combina ambas tecnologías de diseño de sistemas operativos de tiempo real.

Arquitectura

Esta diseñada con una arquitectura dual, por una parte utiliza la arquitectura de microkernel, es decir tiene un segundo kernel, que es el de RTAI, y por la otra, modifica el kernel estándar de Linux para que se convierta en "kernel preemptable".

Características.

- REDICE-Linux proporciona un completo software de desarrollo, y de herramientas para el rápido desarrollo de sistemas de tiempo real embebidos.
- Reloj con alta precisión del orden de microsegundos.
- Sistema de planificación potente y predecible.
- Mecanismo para garantizar rendimiento a las tareas.
- Potencia y fiabilidad del código abierto de Linux.
- Sistema de tiempo real con capacidad de garantizar a las aplicaciones rendimiento y ejecución de tareas con latencias del rango de los microsegundos.

4.8.10 TimeSys Linux

TimeSys Linux es más que una distribución de un kernel de tiempo real, es un conjunto completo de herramientas para el desarrollo de aplicaciones de tiempo real de forma cómoda y rápida. Es una distribución para sistemas embebidos.

Arquitectura

TimeSys Linux se basa en la arquitectura de "kernel preemptable" para proporcionar rendimiento predecible y latencia limitada.

Características.

- En el peor de los casos, tiempo de respuesta de 100 microsegundos.
- Soporte en el mismo sistema, de tiempo real estricto y tiempo real flexible.
- Soporte para aplicaciones Linux estándar y con interfaces POSIX, así como relojes con resolución de microsegundos (dependiendo del hardware).

4.8.11 Linux-SRT

Linux-SRT es una extensión al kernel de Linux que inició el camino hacia la ejecución de aplicaciones de tiempo real. El kernel estándar de Linux no era un sistema operativo multimedia, no garantizaba que el audio, el vídeo u otro proceso crítico ejecutará con una transferencia fijada, por eso nació este proyecto con el objetivo de crear un sistema operativo multimedia que proporcionara una calidad de servicio determinada, pero no estricta, de ahí el nombre, Linux-SRT (soft real time). No es adecuado para sistemas críticos. Este proyecto se ha abandonado actualmente debido al laborioso proceso de modificación del kernel.

Arquitectura

La arquitectura de Linux-SRT es similar a la de KURT, es decir, añadir al kernel estándar requerimientos de tiempo real mediante la inclusión de librerías conforme al estándar POSIX 1003.13.

Características.

- Planificación: El paquete correspondiente al planificador permite especificar términos de calidad de servicio, como por ejemplo el uso de un determinado porcentaje máximo a cada tarea de tiempo real. Esto es útil por ejemplo para grabar cd's o escuchar mp3's sin interrupción.
- Gráficos: La asignación de CPU a una tarea no es el único factor que afecta a la velocidad de las aplicaciones, también lo es el servidor X, por eso se modifica para priorizar la renderización de gráficos en función de los parámetros de planificación de cada cliente X.
- Interfaz de usuario: Permite la configuración de parámetros de planificación a nivel de usuario.

4.8.12 QNX

QNX (pronunciado Q.N.X. o Q-nix) es un sistema operativo de tiempo real basado en Unix que cumple con la norma POSIX. Es desarrollado principalmente para su uso en dispositivos embebidos. Desarrollado por QNX Software Systems empresa canadiense. Esta disponible para las arquitecturas x86, MIPS, PowerPC, SH4 (incluida la videoconsola Dreamcast con una versión muy limitada de este), ARM, StrongARM y xScale. [12]

QNX está basado en una estructura de microkernel, que proporciona características de estabilidad avanzadas frente a fallos de dispositivos, aplicaciones, etc. Photon o Photon microGUI es el sistema de ventanas (servidor y cliente) de QNX, aunque también funciona una versión X Windows.

Está orientado a su utilización en microcontroladores y sistemas críticos.

Arquitectura.

QNX proporciona multitarea, planificación por prioridades con desalojo, y rápido cambio de contexto.

QNX también es notablemente flexible. Desarrolladores pueden personalizar el sistema operativo fácilmente para satisfacer las necesidades de su aplicación. Desde la configuración del kernel en unos módulos pequeños, a un sistema de red amplia para servir cientos de usuarios.

QNX logra su único grado de eficacia, modularidad, y simplicidad a través de dos principios fundamentales:

- La arquitectura del microkernel. Consiste en un kernel pequeño a cargo de un grupo de procesos cooperativos

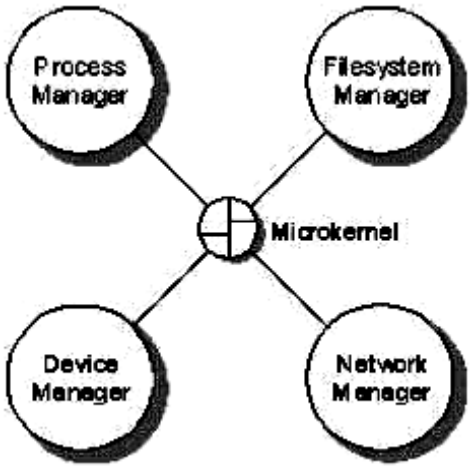

Figura 21: Arquitectura de QNX

- La comunicación entre procesos basada en mensajes.

El microkernel de QNX es responsable de lo siguiente:

- IPC. El microkernel supervisa el ruteo de mensajes; también maneja otras dos formas de IPC: proxies y señales.
- La comunicación de la red a bajo nivel. El microkernel entrega todos los mensajes destinados a los procesos en otros nodos.
- Scheduling de procesos. El scheduler del microkernel decide qué proceso se ejecutará luego.
- Manejo de interrupciones del primer nivel. Todas las interrupciones de hardware y las fallas se rutean primero a través del microkernel, luego se pasa al driver o al administrador del sistema.

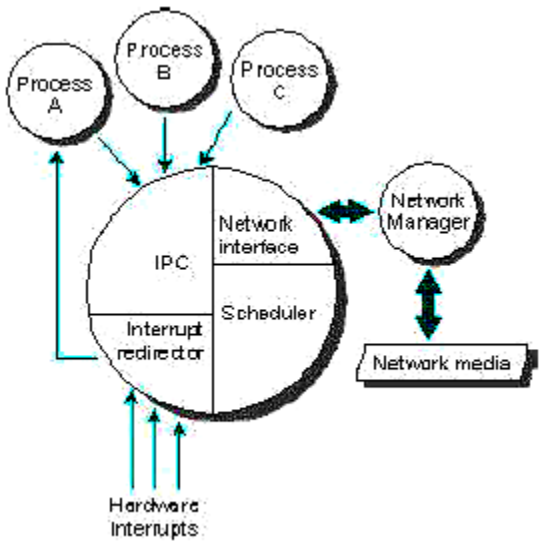

Figura 22: Funciones del MicroKernel

El microkernel de QNX apoya tres tipos esenciales de IPC (comunicación entre procesos): mensajes, proxies y señales

- Mensajes. La forma fundamental de IPC en QNX. Ellos proporcionan la comunicación síncrona entre procesos cooperativos dónde el proceso que envía el mensaje requiere acuse de recibo y potencialmente una contestación al mensaje.
- Proxies. Una forma especial de mensaje. Están especialmente preparados para notificación de eventos dónde el proceso que lo envía no necesita actuar recíprocamente con el destinatario.
- Señales. Una forma tradicional de IPC. Se utilizan para apoyar la comunicación entre procesos de forma asíncrona.

Características.

QNX soporta los estándares API, estándares de utilitarios y la mayoría de los estándares de tiempo real, conocidos como 1003.1, 1003.2 y 1003.3 respectivamente. QNX no soporta algunos estándares en el cual su inclusión provocaría una limitación en la potencia del SO. QNX puede compilarse en POSIX sin forzarlo a perder sus subyacentes características de alta performance. Esto es posible porque POSIX describe la interfase,

pero no tiene requerimientos en cuanto a su implementación. Esto resultó ser una brillante maniobra. Manteniendo su núcleo subyacente de pasaje de mensajes, QNX mantiene sus características de performance en tiempo real.

Para satisfacer las necesidades de varias aplicaciones, QNX proporciona tres métodos de planificación:

- FIFO. Un proceso seleccionado para correr continúa ejecutándose hasta que: voluntariamente abandona el control (por ejemplo hace cualquier llamada al kernel)o bien porque es desalojado por un proceso de prioridad superior. Dos procesos que corren a la misma prioridad pueden usar FIFO para asegurar la exclusión mutua a un recurso compartido. Ningún proceso será desalojado por el otro mientras se está ejecutando.

- Round-robin. Un proceso seleccionado para correr continúa ejecutando hasta que: voluntariamente abandona el control, es desalojado por un proceso de prioridad superior o consume su quantum4

- Adaptativa. Un proceso se comporta de la siguiente forma: Si el proceso consume su quantum (es decir no bloquea), su prioridad está reducida por 1. Esto se conoce como el decaimiento de prioridad. Notar que un el proceso "descendente" no continuará disminuyendo, aun cuando consume otro quantum sin bloquear; dejará caer la prioridad sólo un nivel por debajo de su prioridad original. Y si por el contrario el proceso bloquea, revierte inmediatamente a su prioridad original. Usted puede usar la planificación adaptativa en ambientes dónde los procesos son del tipo background en su mayoría y están compartiendo la computadora con los usuarios interactivos. La planificación adaptativa es el método de planificación predefinido para programas creados por la SHELL.

La latencia de la interrupción es el tiempo de la recepción de una interrupción del hardware hasta que se ejecuta la primera instrucción de un manipulador de interrupción

de software. QNX deja las interrupciones habilitadas casi todo el tiempo, para que la latencia de interrupción sea insignificante. Pero ciertas secciones críticas de código requieren que se desactive temporalmente las mismas. El máximo tiempo de desactivación de las interrupciones define la latencia de interrupción del peor caso, en QNX es muy pequeño.

La latencia de interrupción del peor caso será el tiempo más largo en que QNX desactiva las interrupciones de CPU. La tabla 7 muestra la latencia de interrupción típica cronometrada (Til) para un rango de procesadores:

Latencia de interrupciones	Procesador:
3.3 microsec	166 Pentium de MHz
4.4 microsec	100 Pentium de MHz
5.6 microsec	100 MHz 486DX4
22.5 microsec	33 MHz 386EX

Tabla 7: Latencias de Interrupciones

La latencia del scheduling es el tiempo entre la terminación del manipulador de interrupción y la ejecución de la primera instrucción de un proceso del driver. Esto significa el tiempo que toma cambiar el contexto del proceso actualmente ejecutando y restaurar el contexto del proceso del driver requerido. Aunque es más grande que la latencia de la interrupción, este tiempo también es pequeño en un sistema de QNX.

Otra de las características de QNX es su sistema de ventanas llamado Photon MicroGUI, es escalable y es capaz de correr con menos de 500K de memoria y sin embargo entrega toda la funcionalidad esperada de un sistema de ventanas e introduciendo muchas opciones de conectividad.

Latencia de *scheduling*	Procesador:
4.7 microsec	166 Pentium de MHz
6.7 microsec	100 Pentium de MHz
11.1 microsec	100 MHz 486DX4
74.2 microsec	33 MHz 386EX

Tabla 8: Latencias de Scheduling

Photon microGUI ha sido diseñado para entregar un sistema gráfico de ventanas a ambientes con restricciones de recursos. A través de su única arquitectura, también provee:

- Opciones de escalabilidad. Photon puede ser escalado en un sistema gráfico completo para escritorio. Photon puede ser integrado con sistemas de ventanas heterogéneos.
- Tolerancia a fallos. Si una parte del sistema de ventanas falla, ese componente puede ser reiniciado sin colgar el sistema entero.
- Un microkernel gráfico. Siguiendo la arquitectura del microkernel de QNX, y teniendo este una intercomunicación de proceso muy eficiente, el microkernel gráfico se estructura como un proceso con un equipo de procesos adicionales cooperando con este, comunicándose vía IPC. En la figura 23 podemos ver la arquitectura de este sistema de ventanas.

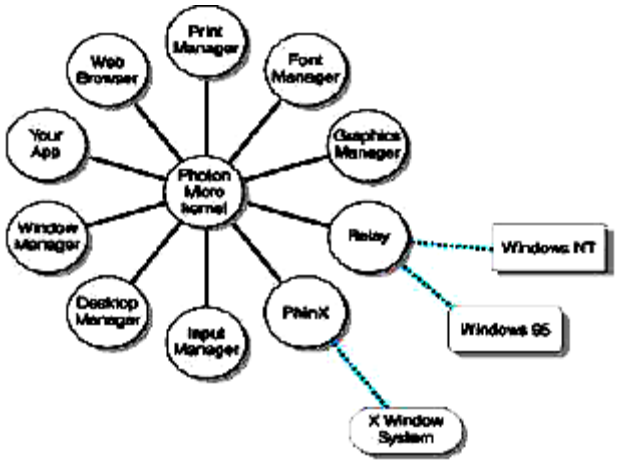

Figura 23: Photon microGUI kernel y procesos cooperadores

4.8.13 VxWorks

Es un componente run-time de la plataforma de desarrollo de sistemas empotrados Tornado II (Windows y UNIX).

Hay dos versiones de VxWorks: VxWorks a secas, que es de la que hablamos aquí. Y VxWorks AE, que está diseñado especialmente para alta disponibilidad con la ayuda distribuida de la mensajería y de la alta tolerancia.

Características.

- Herramientas de desarrollo cruzado host-target:
  - Compilador, Linkador y Cargador para el sistema target.
  - Depurador remoto.
  - Determinación de tiempos de ejecución.
  - Simulador del sistema sobre el host.
- Soporte de un gran número de placas comerciales mediante Board Support Packages (más de 200), así como de placas a medida:
  - Inicialización del HW.
  - Configuración de interrupciones y temporizadores
  - Memory mapping, etc.

Consta de una interfaz con varias APIs (más de 1800 funciones).

Es utilizado en diferentes tipos de aplicaciones: desde automoción (Antilock Braking Systems) a aplicaciones espaciales (Mars PathFinder), equipamiento de oficina (impresoras, faxes, etc.) y electrodomésticos (microondas, lavavajillas, etc.).

Está adaptado a diferentes arquitecturas de CPU’s tales como: PowerPC, Intel 80x86, Motorola 68K, Intel 80960, ARM, SPARC, etc.

Arquitectura.

La arquitectura está basada en microkernel, dicho microkernel lo han llamado wind de unos pocos KB que incluye:

- Planificador multitarea basado en prioridades y con desplazamiento, el cual consta de 256 prioridades y tiene limitado el número de tareas.
- Primitivas de sincronización y comunicación entre tareas.
- Soporte a interrupciones.
- Gestión de watchdog timers.
- Gestión de memoria.

Capítulo 5

RTAI

Este capítulo se describe las principales características y arquitectura de RTAI, ya que bajo este sistema se ha construido la plataforma de tiempo real para el PC104.

5.1 Introducción

RTAI es una implementación de Linux para tiempo real basada en RTLinux. Añade un pequeño kernel de tiempo real bajo el kernel estándar de Linux y trata al kernel de Linux como una tarea con la menor prioridad. RTAI además proporciona una amplia selección de mecanismos de comunicación entre procesos y otros servicios de tiempo real.

Adicionalmente, RTAI proporciona un módulo llamado LXRT para facilitar el desarrollo de aplicaciones de tiempo real en el espacio de usuario.

5.2 Arquitectura

RTAI tiene una arquitectura similar a RTLinux. Al igual que RTLinux, RTAI trata el kernel estándar de Linux como una tarea de tiempo real con la menor prioridad, lo que hace posible que se ejecute cuando no haya ninguna tarea con mayor prioridad ejecutándose. Las operaciones básicas de las tareas de tiempo real son implementadas como módulos del kernel al igual que RTLinux. RTAI maneja las interrupciones de

periféricos y son atendidas por el kernel de Linux después de las posibles acciones de tiempo real que hayan podido ser lanzadas por efecto de la interrupción.

La siguiente figura muestra la arquitectura básica de RTAI, que es muy similar a la de RTLinux. Las interrupciones se originan en el procesador y en los periféricos. Las originadas en el procesador (principalmente señales de error como división por cero), son manejadas por el kernel estándar, pero las interrupciones de los periféricos (como los relojes) son manejadas por RTAI Interrupt Dispatcher. RTAI envía las interrupciones a los manejadores del kernel estándar de Linux cuando no hay tareas de tiempo real activas. Las instrucciones de activar/desactivar las interrupciones del kernel estándar son reemplazadas por macros que se enlazan con las instrucciones de RTAI. Cuando las interrupciones están desactivadas en el kernel estándar, RTAI encola las interrupciones para ser repartidas después de que el kernel estándar haya activado las interrupciones de nuevo. [11]

Adicionalmente, se puede ver en la figura el mecanismo de comunicación entre procesos (IPC), que esta implementado de forma separado por Linux y por RTAI. También existe un planificador (scheduler) distinto para Linux y para RTAI.

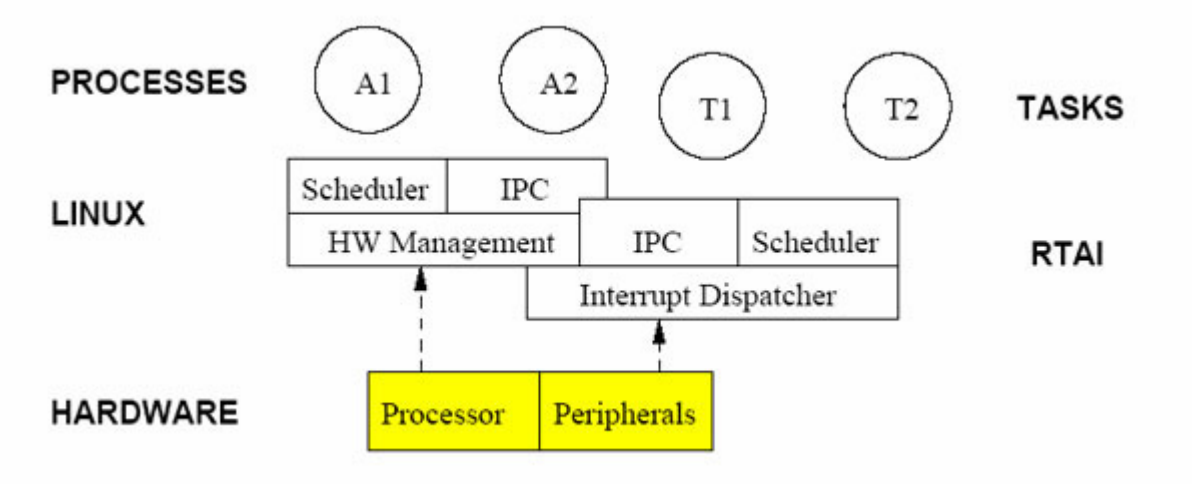

Figura 24: Arquitectura de RTAI

5.2.1 HAL - Hardware Abstraction Layer

Los desarrolladores de RTAI introducen el concepto de Real Time Hardware Abstraction Layer (RTHAL) que es usado para interceptar las interrupciones hardware y procesarlas después. RTHAL es una estructura instalada en el kernel de Linux que reúne los punteros a los datos internos del hardware relacionados en el kernel y las funciones necesarias por RTAI para operar.

El objetivo de RTHAL es minimizar el número de cambios necesarios sobre el código del kernel y por tanto mejorar el mantenimiento de RTAI y del código del kernel de Linux. Con RTHAL, las diferentes operaciones (ej. manejar interrupciones) son más fáciles de cambiar ó modificar sin tener que interferir con la implementación de Linux. Por ejemplo, la estructura de RTHAL contiene la tabla de manejadores de interrupción, la cual es una lista de las funciones que son llamadas para manejar las diferentes interrupciones. El cambio consiste únicamente en modificar unas 20 líneas del código del kernel de Linux y añadir unas 50 nuevas líneas. [11]

Cuando RTHAL es instalado en Linux, las funciones y las estructuras de datos relacionada con la interacción con el hardware son reemplazadas por punteros a la estructura de RTHAL.

5.2.2 Planificación

La unidad de planificación de RTAI es la tarea. Siempre hay al menos una tarea, llamada kernel de Linux, que ejecuta como la tarea de menor prioridad. Cuando las tareas de tiempo real son añadidas, el planificador da entonces mayor prioridad a éstas sobre la tarea del kernel de Linux. El planificador proporciona servicios tales como suspend, resume, yield, make periodic, wait until, que son usadas en varios sistemas operativos de tiempo real.

El planificador es implementado como un módulo del kernel dedicado (contrario a RTLinux) lo que facilita la implementación de planificadores alternativos si es necesario. Actualmente hay tres tipos de planificadores dependiendo del tipo de máquina:

- Uniprocesador (UP)
- Multiprocesador simétrico (SMP)
    - Esta diseñado para maquinas SMP y proporciona un interfaz para las aplicaciones de forma que es posible seleccionar el procesador ó procesadores que deben ejecutar una tarea. Si el usuario no especifica un procesador para la tarea, SMP selecciona el procesador en función de la carga de trabajo.
- Multi-Uniprocesador (MUP)
    - Puede ser usado con ambos, pero al contrario que SMP, a las tareas se les debe especificar el procesador que deben usar. Viéndolo por el lado positivo, el planificador MUP permite un mecanismo de tiempo más flexibles para las tareas que los planificadores SMP ó UP.

5.3 Características

Con el objetivo de hacer el desarrollo de aplicaciones más fáciles y flexibles posibles, los desarrolladores de RTAI han introducido diferentes mecanismos para la comunicación entre procesos (IPC), entre las tareas de tiempo real y los procesos en el espacio de usuarios. Como añadido al mecanismo IPC, RTAI proporciona servicios de manejo de memoria y threads compatibles con Posix. [11]

5.3.1 Comunicación entre procesos (IPC - Inter-process comunication)

RTAI proporciona una variedad de mecanismos para la comunicación entre procesos. Aunque los sistemas Unix proporcionan mecanismos similares a IPC para los procesos en el espacio de usuario, RTAI necesita proporcionar una implementación propia para que las tareas de tiempo real puedan usar este mecanismo y no usen el estándar del kernel de Linux. Los diferentes mecanismos de IPC están incluidos como módulos de kernel, lo que facilita la carga cuando son necesarios. Como ventaja adicional el uso de módulos para los servicios, IPC facilita el mantenimiento y la expansión.

El antiguo mecanismo básico de comunicación de RTAI eran los FIFOs. FIFO es un asíncrono y no bloqueante canal de comunicación entre los procesos de Linux y las tareas de tiempo real. La implementación de RTAI de FIFO esta basado en la implementación de RTLinux, pero RTAI proporciona algunas características que no son posibles en RTLinux. Primeramente RTAI puede lanzar señales cuando hay eventos en el FIFO (escritura de nuevos datos). Los procesos en el espacio de usuario pueden entonces crear un manejador para la señal por los mecanismos estándar de Unix. Sin embargo, este mecanismo no es necesario para los procesos de usuario que quieran leer ó escribir del FIFO. Adicionalmente, puede haber múltiples lectores y escritores en el FIFO, cosa que no es posible en la versión de RTLinux. [7]

Finalmente, los identificadores FIFO pueden ser dinámicamente localizados con un nombre simbólico. Antes era necesario establecer un identificador global para el FIFO, lo que causaba problemas cuando múltiples e independientes procesos y tareas lo usaban.

Los semáforos es otra herramienta básica de sincronización entre procesos usada en los sistemas operativos. RTAI proporciona un API para usar semáforos, aunque cada semáforo esta técnicamente asociado a un FIFO, por tanto cada semáforo usa una entrada global del FIFO. Como añadido al servicio básico de semáforos, un semáforo

puede estar asociado con un reloj, el cual puede ser usado para despertar un proceso encolado en un semáforo, incluso cuando el semáforo aun esta cerrado.

La memoria compartida proporciona una alternativa a IPC y al paradigma FIFO cuando un modelo de comunicación diferente es requerido. La memoria compartida es un bloque común de memoria que puede ser leído ó escrito por un proceso y una tarea en el sistema. Como los diferentes procesos pueden operar de forma asíncrona en la región de memoria, es necesario un diseño para asegurar que los datos no sean sobrescritos de forma intencionada. [7]

Finalmente, el método más flexible de IPC quizás sean los mailboxes. Cualquier número de procesos pueden enviar y recibir mensaje de y desde un mailbox. Un mailbox almacena mensajes hasta un límite que se defina, y contrario a los FIFOs, mailbox preserva los mensajes que están en el límite. Puede haber un número arbitrario de mailbox activos en el sistema simultáneamente. RTAI también facilita la comunicación entre procesos mediante RPC.

5.3.2 Gestión de memoria

En las primeras versiones de RTAI la memoria tenía que ser asignada estáticamente y no era posible la asignación en tiempo real. Sin embargo en las actuales versiones se incluye un módulo gestor de memoria que permite la asignación dinámica de memoria por parte de las tareas de tiempo real usando un interfaz basado en una librería estándar de C.

RTAI preasigna trozos de memoria antes de la ejecución de tiempo real. Cuando la tarea de tiempo real llama a la función rt_malloc(), la respuesta que obtiene es el trozo preasignado. Antes de que el espacio se agote, RTAI reserva nuevos trozos de memoria (preasigna) para futuras llamadas. De manera similar ocurre con la función rt_free(), en este caso, se libera la memoria preasignada a la espera de futuras reservas. Cuando la

suma de memoria liberada es mayor que un valor para una marca, se ordena su liberación.

5.3.3 Threads Posix

RTAI tiene módulos que proporcionan la implementación de threads de acuerdo al estándar POSIX 1003.1c. Usando las operaciones especificadas en el estándar, el usuario puede manejar de manera similar a como lo hace con los threads posix convencionales, excepto en cuanto a los conceptos de joining y detaching.

Los threads de un mismo proceso comparten el espacio de memoria, por tanto es fácil el intercambio de información entre ellos, sin embargo, el uso de áreas memoria compartida son necesarias para la sincronización. También se proporcionan los mecanismos de mutex y de variables de condición. [7]

5.3.4 LXRT: User-space Interface to RTAI

LXRT es un API para RTAI que hace posible el desarrollo de aplicaciones de tiempo real en el espacio de usuario sin tener que crear módulos para el kernel. Esto es útil en primer lugar porque el espacio de memoria destinado al kernel no esta protegido de accesos inválidos, lo que puede provocar la corrupción de datos y el mal funcionamiento del kernel de Linux. En segundo lugar, si el kernel es actualizado, los módulos necesitan ser recompilados lo que puede provocar que sean incompatibles con la nueva versión.

Mientras se desarrolla una aplicación de tiempo real en el espacio de usuario el programador puede usar las herramientas estándar de depuración hasta que ya no contienen errores, pero en este caso se trata de procesos de tiempo real flexibles (soft). Además, los servicios proporcionados por las llamadas al sistema de Linux están disponibles para las tareas. La ventaja de esto es que cuando la aplicación esté libre de errores puede convertirse a un módulo en el espacio del kernel, por lo que ya tendremos

una tarea de tiempo real estricto. Sin embargo, al hacer esto, las llamadas al sistema utilizadas no sirven y deberán ser cambiadas por las proporcionadas por RTAI.

El cambio de la aplicación de procesos del espacio de usuario a tareas de tiempo real es fácil porque LXRT proporciona un API simétrico para la comunicación entre procesos y otros servicios de RTAI, lo que significa que el mismo API puede ser usado por las tareas de tiempo real y por los procesos del espacio de usuario. El mismo API de LXRT puede ser también usado cuando 2 procesos del espacio de usuario ó 2 tareas de tiempo real se comunican entre si, esto implica que varios relojes y mensajes del sistema proporcionados por LXRT puedan ser usados incluso cuando la aplicación no requiera tiempo real.

LXRT permite a las aplicaciones el intercambio dinámico entre tiempo real flexible y estricto mediante el uso de una simple llamada en el espacio de usuario. Cuando una aplicación esta en el modo de tiempo real flexible, usa el planificador de Linux, pero requiere el uso de la política de planificación FIFO. Sin embargo la planificación de procesos FIFO puede provocar la perdida de ranuras de tiempo por varias razones, por ejemplo, porque se esta ejecutando un manejador de interrupciones ó el planificador de tareas de RTAI.

Con el objetivo de facilitar el paso a tiempo real estricto, LXRT crea un agente en el espacio del kernel para cada proceso del espacio de usuario con requerimientos de tiempo real. Cuando el proceso entra en modo de tiempo real estricto, el agente desactiva las interrupciones, y mueve al proceso fuera del planificador de Linux y lo añade a la cola del planificador de RTAI. Ahora el proceso no es interrumpido por las interrupciones de otro proceso Linux. Para poder realizar este cambio el proceso debe asegurar que la memoria usada por el proceso este en la memoria RAM y debe desactivar la paginación usando la función mlockall(). Esta llamada no debe ser usada en modo tiempo real estricto. [11]

Aunque los desarrolladores de RTAI ven bien el desarrollo de aplicaciones de tiempo real estricto usando LXRT, el tiempo de respuesta no es tan bueno como la ejecución de las tareas como módulos del kernel.

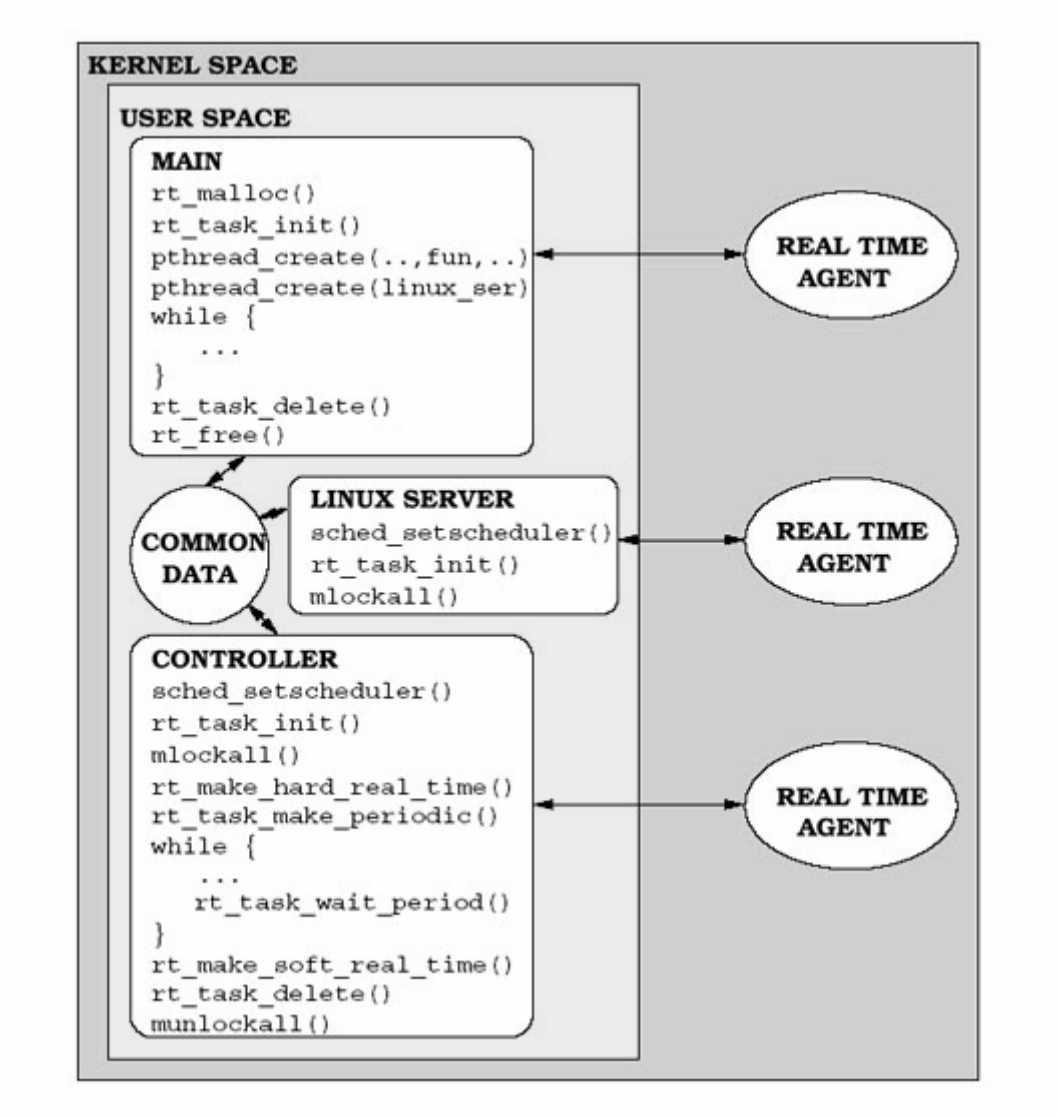

Figura 25: Arquitectura de una aplicación simple de tiempo real

RTAI puede construirse sobre un kernel de Linux bajo dos enfoques diferentes, uno utilizando RTHAL (MicroKernel) y otro utilizando ADEOS (NanoKernel), por problemas de patentes la plataforma de tiempo real ha sido construida utilizando el enfoque de ADEOS, cuyas características se describen a continuación.

5.4 ADEOS

El objetivo de ADEOS es proporcionar un entorno flexible para compartir los recursos hardware para múltiples sistemas operativos ó múltiples instancias de un mismo sistema operativo. [12]

ADEOS activa múltiples kernels, llamados dominios, que existen simultáneamente sobre el mismo hardware. Ninguno de estos dominios necesariamente conoce la existencia del resto, pero todos ellos si conocen de la existencia de ADEOS. Un dominio puede ser un Sistema Operativo completo, pero no necesariamente.

Actualmente ADEOS permite compartir las interrupciones hardware.

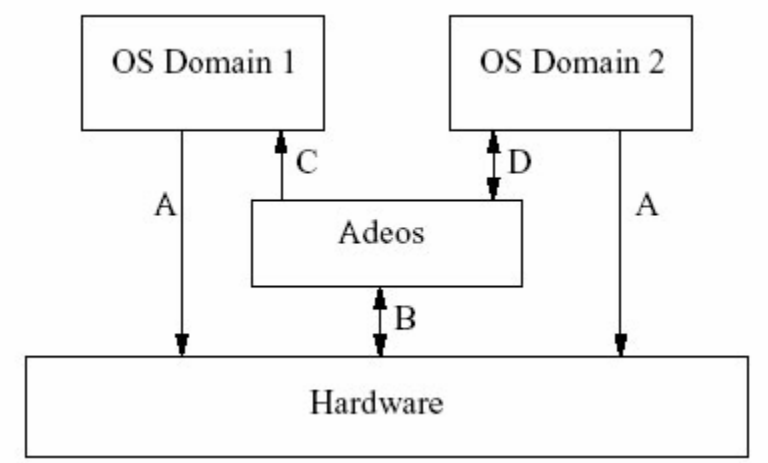

Figura 26: Interrupciones en ADEOS

5.4.1 Arquitectura

La arquitectura de ADEOS es la de kernel dual y más específicamente la llamada Nanokernel.

5.4.1.1 Dominios y Pipelines

Para permitir que las interrupciones y los eventos del sistema sean repartidos para compartir por los múltiples kernels (normalmente SO completos), ADEOS define lo que ha llamado abstractamente "dominio".

Un dominio es un componente software del kernel base al cuál ADEOS puede notificar:

- Las interrupciones hardware.
- Llamadas al sistema de las aplicaciones Linux.
- Eventos del sistema lanzados por el kernel de Linux.
- Otros eventos que personalicemos nosotros.

Un dominio puede ser accesible como un módulo dinámico del kernel, ó como uno estático formando parte de una imagen del kernel, no produciendo ningún tipo de incidencia en el comportamiento de ADEOS.

ADEOS también puede asegurar que los eventos sean disparados en el orden correcto para los distintos dominios definidos, gracias a ello, es posible proporcionar determinismo.

Esto nos permite asignar a cada dominio una prioridad estática. El valor de esta prioridad determina el orden en que los eventos son tratados por los dominios. Todos los dominios son encolados de acuerdo a sus respectivas prioridades, formando un pipeline de forma abstracta, que es el que usa ADEOS para manejar el flujo de eventos desde el más prioritario hacia el menos prioritario. Los eventos de entrada son encauzados a la cabecera del pipeline (dominio más prioritario) y progresan hacia la cola (dominio menos prioritario).

El código del kernel de Linux es enteramente el sólo un dominio especial predefinido, que es creado por ADEOS en las primeras etapas de carga de Linux. Este dominio

inicial normalmente hace referencia al "root" domain hasta que la infraestructura proporciona lo suficiente para cargar el resto de dominios dinámicamente (ej. el módulo cargador del kernel)

5.4.1.2 Modo de trabajo

ADEOS controla todas las interrupciones, los traps de la CPU y las excepciones del nivel hardware, ya que reprograma el software del manejador. De hecho, ADEOS actualmente reemplaza los manejadores instalados anteriormente por Linux durante el arranque por los suyos propios para las correspondientes interrupciones y eventos.

En la plataforma x86, se hace simplemente cambiando el descriptor de interrupción de la tabla.

En cualquier momento dada una CPU con ADEOS activado, un dominio puede estar:

- Ejecutándose.
- Interrumpido por un dominio más prioritario durante el procesamiento de una interrupción/evento de entrada mientras él estaba procesando un evento pendiente.
- Voluntariamente autosuspendido, después de que todas las interrupciones/eventos hayan sido procesados.

Cuando un dominio ha finalizado de procesar una interrupción/evento que ha recibido, llama a un servicio especial de ADEOS (adeos_suspend_domain()) el cual provoca que se pase la interrupción/evento al siguiente dominio del pipeline y así sucesivamente, realizándose un ciclo del más prioritario al menos prioritario.

Hay que destacar que ADEOS reanuda un dominio sólo si tiene que procesar una interrupción/evento ó si ocurre que ha sido interrumpido por un dominio más prioritario que ha recibido alguna interrupción/evento para procesar, en otras palabras, no se

produce intercambio entre dos dominios a menos que haya trabajo pendiente para alguno de ellos.

Cuando una interrupción ocurre en un sistema ocioso, ADEOS despierta al dominio más prioritario interesado en ella y el ciclo de proceso del pipeline comienza de nuevo.

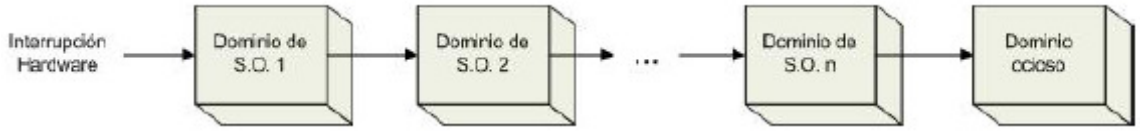

Figura 27: Interrupciones y Dominios

5.4.2 ¿Por qué necesitamos ADEOS?

- Porque se necesita controlar determinísticamente el flujo de interrupciones hardware usando una capa software antes de que el kernel de Linux las procese. Este control incluye la interceptación, el enmascarado y/o la priorización de las interrupciones.
- Porque se necesita monitorizar las llamadas al sistema de Linux, añadiendo más información y sin tener que modificar el código de las llamadas al sistema.
- Porque se quiere un mecanismo para monitorizar los eventos internos que ocurren en el kernel de Linux como pueden ser la planificación de tareas, la creación de procesos ó las señales que capturan las tareas.

Capítulo 6

Construcción de La Plataforma de Tiempo Real

En este capítulo se construye un sistema operativo basado en un kernel de Linux, con atributos para procesar tares en tiempo real, sobre el PC104

6.1 Introducción

El sistema se construirá utilizando una distribución Linux ya instalada (Fedora Core 3). Este sistema Linux existente (el anfitrión) se utilizará como punto de inicio para suministrar los programas necesarios, como un compilador, un enlazador y un intérprete de comandos, para construir el nuevo sistema.

El presente capítulo para la construcción de un nuevo sistema, esta compuesto por 7 secciones.

La primera sección describe cómo crear una nueva partición nativa Linux y un sistema de ficheros, el sitio donde se compilará e instalará el nuevo sistema.

La segunda sección explica qué paquetes y parches deben descargarse para construir un nuevo sistema y cómo guardarlos en el nuevo sistema de ficheros.

La tercera sección muestra cómo configurar un entorno de trabajo adecuado.

La cuarta sección describe la instalación de una serie de paquetes que formarán el entorno básico de desarrollo (o herramientas principales) utilizado para construir el sistema real.

En la quinta sección se construye el auténtico sistema.

En la sexta sección se configuran los guiones de arranque.

En la séptima sección se configuran el núcleo y el gestor de arranque.

Tras completar los pasos de este capítulo, el PC/104 estará listo para iniciar dentro del nuevo sistema construido.

6.2 Sección 1

En esta sección se preparará la partición que contendrá el sistema. Se creará la propia partición, se creará un sistema de ficheros en ella y se montará.

6.2.1 Crear una nueva partición [10]

Se inicia el programa de particionado **fdisk** pasándole como argumento el nombre del disco duro en el que debe crearse la nueva partición, en este caso /dev/sda1 (la compact flash es reconocida como /dev/sda) y luego se crea la nueva partición de tipo Linux.

```
fdisk /dev/sda

Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
```

```
p
Partition number (1-4): 1
First cylinder (1-1015, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-1015, default 1015):
Using default value 1

Command (m for help): p

Disk /dev/sda: 2 heads, 62 sectors, 1015 cylinders
Units = cylinders of 124 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/sda1             1      1015     62899   83  Linux
```

Se activa la partición:

```
Command (m for help): a
Partition number (1-4): 1

Command (m for help): p

Disk /dev/sda: 2 heads, 62 sectors, 1015 cylinders
Units = cylinders of 124 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/sda1   *         1      1015     62899   83  Linux
```

Finalmente se guardan las modificaciones realizadas:

```
Command (m for help): w
The partition table has been altered!
```

6.2.2 Crear un sistema de ficheros en la partición

Ahora que hay preparada una partición en blanco ya puede crearse el sistema de ficheros. El más usado en el mundo de Linux es el llamado “second extended file system” (segundo sistema de ficheros extendido, o ext2). Crearemos un sistema de ficheros ext2. [10]

Para crear un sistema de ficheros ext2 en la partición creada, se ejecuta lo siguiente:

mke2fs -v /dev/sda1

6.2.3 Montar la nueva partición

Ahora que se ha creado un sistema de ficheros es necesario hacer accesible la partición. El sistema de ficheros se decide montar en /mnt/lfs.

El punto de montaje es asignado a la variable de entorno LFS ejecutando:

export LFS=/mnt/lfs

Se crea el punto de montaje y se monta el sistema de ficheros LFS ejecutando:

mkdir -pv $LFS

mount -v /dev/sda1 $LFS

6.3 Sección 2

6.3.1 Paquetes y parches

Es necesario guardar todos los paquetes y parches descargados en algún sitio que esté disponible durante toda la construcción. También se necesita un directorio de trabajo en el que desempaquetar las fuentes y construirlas. Puede usarse $LFS/sources tanto para almacenar los paquetes y parches como directorio de trabajo. Al usar este directorio, los

elementos requeridos se encontrarán en la partición creada y estarán disponibles durante todas las fases del proceso de construcción. [10]

Para crear este directorio, se ejecuta el siguiente comando como usuario root

mkdir -v $LFS/sources

6.3.2 Todos los paquetes

La siguiente lista muestra todos los paquetes utilzazos para construir el sistema.

- **Autoconf (2.59) - 908 kilobytes (KB)**
- **Automake (1.9.5) - 748 KB**
- **Bash (3.0) - 1,824 KB**
- **Bash Documentation (3.0) - 1,994 KB**
- **Binutils (2.15.94.0.2.2) - 11,056 KB**
- **Bison (2.0) - 916 KB**
- **Bzip2 (1.0.3) - 596 KB**
- **Coreutils (5.2.1) - 4,184 KB**
- **DejaGNU (1.4.4) - 852 KB**
- **Diffutils (2.8.1) - 648 KB**
- **E2fsprogs (1.37) - 3,100 KB**
- **Expect (5.43.0) - 416 KB**
- **Findutils (4.2.23) - 784 KB**
- **Flex (2.5.31) - 672 KB**
- **Gawk (3.1.4) - 1,696 KB**
- **GCC (3.4.3) - 26,816 KB**
- **Gettext (0.14.3) - 4,568 KB**

- **Glibc (2.3.4) - 12,924 KB**
- **Glibc-Linuxthreads (2.3.4) - 236 KB**
- **Grep (2.5.1a) - 520 KB**
- **Groff (1.19.1) - 2,096 KB**
- **GRUB (0.96) - 768 KB**
- **Gzip (1.3.5) - 284 KB**
- **Hotplug (2004_09_23) - 40 KB**
- **Iana-Etc (1.04) - 176 KB**
- **Inetutils (1.4.2) - 752 KB**
- **IPRoute2 (2.6.11-050330) - 276 KB**
- **Less (382) - 216 KB**
- **LFS-Bootscripts (3.2.1) - 32 KB**
- **Libtool (1.5.14) - 1,604 KB**
- **Linux (2.6.11.12) - 35,792 KB**
- **Linux-Libc-Headers (2.6.11.2) - 2,476 KB**
- **M4 (1.4.3) - 304 KB**
- **Make (3.80) - 904 KB**
- **Mktemp (1.5) - 68 KB**
- **Module-Init-Tools (3.1) - 128 KB**
- **Module-Init-Tools-Testsuite (3.1) - 34 KB**
- **Ncurses (5.4) - 1,556 KB**
- **Patch (2.5.4) - 156 KB**
- **Perl (5.8.7) - 9,628 KB**
- **Procps (3.2.5) - 224 KB**
- **Readline (5.0) - 1,456 KB**

- **Sed (4.1.4) - 632 KB**
- **Shadow (4.0.9) - 1,084 KB**
- **Sysklogd (1.4.1) - 72 KB**
- **Sysvinit (2.86) - 88 KB**
- **Tar (1.15.1) - 1,580 KB**
- **Tcl (8.4.9) - 2,748 KB**
- **Texinfo (4.8) - 1,492 KB**
- **Udev (056) - 476 KB**
- **Udev Rules Configuration - 5 KB**
- **Util-linux (2.12q) - 1,344 KB**
- **Zlib (1.2.3) - 415 KB**

Aparte de los paquetes, también se necesitan varios parches. Estos parches corrigen pequeños errores en los paquetes que debería solucionar su desarrollador. Los parches también hacen pequeñas modificaciones para facilitar el trabajo con el paquete. Los siguientes parches son necesarios para construir el sistema. [10]

- **Bash Avoid Wcontinued Patch - 1 KB**
- **Bash Various Fixes - 23 KB**
- **Binutils Build From Host Running Gcc4 Patch - 2 KB**
- **Bzip2 Documentation Patch - 1 KB**
- **Bzip2 Bzgrep Security Fixes Patch - 1 KB**
- **Coreutils Suppress Uptime, Kill, Su Patch - 15 KB**
- **Coreutils Uname Patch - 4 KB**
- **Expect Spawn Patch - 7 KB**
- **Flex Brokenness Patch - 156 KB**
- **GCC Linkonce Patch - 12 KB**

- **GCC No-Fixincludes Patch - 1 KB**
- **GCC Specs Patch - 14 KB**
- **Glibc Rtld Search Dirs Patch - 1 KB**
- **Glibc Fix Testsuite Patch - 1 KB**
- **Glibc TLS Assertion Patch - 6 KB**
- **Gzip Security Patch - 2 KB**
- **Inetutils Kernel Headers Patch - 1 KB**
- **Inetutils No-Server-Man-Pages Patch - 4 KB**
- **Mktemp Tempfile Patch - 3 KB**
- **Perl Libc Patch - 1 KB**
- **Readline Fixes Patch - 7 KB**
- **Sysklogd Fixes Patch - 27 KB**
- **Tar Sparse Fix Patch - 1 KB**
- **Texinfo Tempfile Fix Patch - 2 KB**
- **Util-linux Cramfs Patch - 3 KB**
- **Util-linux Umount Patch - 1 KB**

6.4 Sección 3

Todos los programas compilados en la sección 4 se instalarán bajo $LFS/tools para mantenerlos separados de los programas compilados en la sección 5. Los programas compilados aquí son sólo herramientas temporales y no formarán parte del sistema final. Al mantener estos programas en un directorio aparte podrán eliminarse fácilmente tras su uso. Esto también evita que acaben en los directorios de producción del anfitrión.

Se crea el directorio necesario ejecutando lo siguiente como *root*:

mkdir -v $LFS/tools

En el próximo paso es crear un enlace /tools en el sistema anfitrión. Este apuntará al directorio que se acaba de crear en la partición de trabajo. Se ejecuta este comando también como *root*:

ln -sv $LFS/tools /

El enlace simbólico creado posibilita que el conjunto de herramientas se compile siempre en referencia a /tools, de forma que el compilador, ensamblador y enlazador funcionarán en esta sección (en la que todavía se utilizan algunas herramientas del sistema anfitrión) y en la siguiente (cuando "se cambie la raíz" a la partición de trabajo).

6.5 Sección 4

Esta sección muestra cómo compilar e instalar un sistema Linux mínimo. Este sistema contendrá sólo las herramientas necesarias para poder iniciar la construcción del sistema definitivo en la sección 5.

La construcción de este sistema minimalista se hace en dos etapas. La primera es construir un conjunto de herramientas independiente del sistema anfitrión (compilador, ensamblador, enlazador, librerías y unas pocas herramientas útiles). La segunda etapa utiliza estas herramientas para construir el resto de herramientas esenciales.

Los ficheros compilados en esta sección se instalan bajo el directorio $LFS/tools para mantenerlos separados de los ficheros que se instalan en la siguiente sección y de los directorios de producción del anfitrión. Puesto que los paquetes compilados aquí son puramente temporales, no se desea que estos ficheros contaminen el futuro sistema.

6.5.1 Binutils-2.15.94.0.2.2 - Fase 1

El paquete Binutils contiene un enlazador, un ensamblador y otras utilidades para trabajar con ficheros objeto. Es importante que Binutils sea el primer paquete que se compile, pues tanto Glibc como GCC llevan a cabo varias comprobaciones sobre el enlazador y el ensamblador, disponibles para determinar qué características activar.

La documentación de Binutils recomienda construirlo en un directorio dedicado, fuera del árbol de las fuentes:

mkdir -v ../binutils-build
cd ../binutils-build

Se prepara Binutils para su compilación:

../binutils-2.15.94.0.2.2/configure --prefix=/tools --disable-nls

Significado de las opciones de configure:

--prefix=/tools

Esto le indica al guión configure que los programas de Binutils se instalarán en el directorio /tools.

--disable-nls

Esta opción desactiva la internacionalización, pues i18n no es necesario en las herramientas temporales.

Se compila el paquete:

make

Se instala el paquete:

make install

Se prepara el enlazador para la posterior fase de “ajuste”:

make -C ld clean

make -C ld LIB_PATH=/tools/lib

Significado de los parámetros de make:

-C ld clean

Esto le indica al programa make que elimine todos los ficheros compilados que haya en el subdirectorio ld.

-C ld LIB_PATH=/tools/lib

Esta opción vuelve a construir todo dentro del subdirectorio ld. Especificar la variable LIB_PATH del Makefile en la línea de comandos permite obviar su valor por defecto y apuntar al directorio de herramientas temporales. El valor de esta variable especifica la ruta de búsqueda de librerías por defecto del enlazador.

6.5.2 GCC-3.4.3 - Fase 1

El paquete GCC contiene la colección de compiladores GNU, que incluye los compiladores C y C++.

La documentación de GCC recomienda construirlo en un directorio dedicado, fuera del árbol de las fuentes:

mkdir -v ../gcc-build

cd ../gcc-build

Se prepara GCC para su compilación:

**../gcc-3.4.3/configure --prefix=/tools **

**--libexecdir=/tools/lib --with-local-prefix=/tools **

--disable-nls --enable-shared --enable-languages=c

Significado de las opciones de configure:

--with-local-prefix=/tools

Esta opción es para eliminar /usr/local/include de las rutas de búsqueda por defecto de **gcc**. Esto no es esencial, sin embargo ayuda a minimizar la influencia del sistema anfitrión.

--enable-shared

Esta opción permite construir libgcc_s.so.1 y libgcc_eh.a. Tener a libgcc_eh.a disponible asegura que el guión configure de Glibc (el siguiente paquete por compilar) produzca los resultados apropiados.

--enable-languages=c

Esta opción asegura que sólo se construya el compilador de C.

Se compila el paquete:

make bootstrap

Significado de los parámetros de make:

bootstrap

Este objetivo no sólo compila GCC, sino que lo compila varias veces. Usa los programas compilados la primera vez para compilarse a sí mismo una segunda vez y luego una tercera. Después compara la segunda compilación con la tercera para asegurarse que puede reproducirse a sí mismo sin errores. Esto también implica que se ha compilado correctamente.

Se instala el paquete:

make install

6.5.3 Linux-Libc-Headers-2.6.11.2

El paquete Linux-Libc-Headers contiene las cabeceras del núcleo. Durante años ha sido una práctica común utilizar las llamadas cabeceras “crudas” del núcleo (extraídas de un paquete del núcleo) en /usr/include, pero en el transcurso de los últimos años los desarrolladores del núcleo han tomado la firme postura de que esto no debe suceder. Así nació el proyecto Linux-Libc-Headers, destinado a mantener una versión estable de la API de las cabeceras Linux.

Se instala los ficheros de cabecera:

cp -Rv include/linux /tools/include

cp -Rv include/linux /tools/include

6.5.4 Glibc-2.3.4

El paquete Glibc contiene la librería C principal. Esta librería proporciona todas las rutinas básicas para la ubicación de memoria, búsqueda de directorios, abrir y cerrar ficheros, leerlos y escribirlos, manejo de cadenas, coincidencia de patrones, aritmética, etc.

La documentación de Glibc recomienda construirlo fuera del árbol de las fuentes, en un directorio de construcción dedicado:

mkdir -v ../glibc-build

cd ../glibc-build

Se prepara Glibc para su compilación:

```
../glibc-2.3.4/configure --prefix=/tools \
      --disable-profile --enable-add-ons \
```

**--enable-kernel=2.6.0 --with-binutils=/tools/bin **
**--without-gd --with-headers=/tools/include **
--without-selinux

Significado de las opciones de configure:

--disable-profile

Esto construye las librerías sin información de perfiles.

--enable-add-ons

Esto le indica a Glibc que utilice el añadido NPTL como su librería de hilos.

--enable-kernel=2.6.0

Esto le indica a Glibc que compile la librería con soporte para núcleos Linux 2.6.x.

--with-binutils=/tools/bin

Aunque no es necesario, esta opción asegura que no haya equívocos sobre qué programas de Binutils se utilizarán durante la construcción de Glibc.

--without-gd

Esto evita la construcción del programa **memusagestat**, el cual insiste en enlazarse contra las librerías del sistema anfitrión (libgd, libpng, libz y demás).

--with-headers=/tools/include

Esto le indica a Glibc que se compile contra las cabeceras recién instaladas en el directorio de herramientas, para que conozca exactamente las características que tiene el núcleo y pueda optimizarse correctamente.

--without-selinux

Cuando se construye a partir de un anfitrión que utiliza la funcionalidad de SELinux (como Fedora Core 3), Glibc se construirá con soporte para SELinux.

Como las herramientas del entorno no contienen soporte para SELinux, una Glibc compilada con dicho soporte no funcionará correctamente.

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.5 Ajustar las herramientas

Ahora que se han instalado las librerías de C temporales, todas las herramientas que se compilen en el resto de esta sección deberían enlazarse contra ellas. Para conseguirlo, deben ajustarse el enlazador y el fichero specs del compilador.
El enlazador se instala ejecutando el siguiente comando desde el directorio binutils-build:

make -C ld install

Desde ahora todo se enlazará solamente contra las librerías que hay en /tools/lib.

Lo siguiente es corregir el fichero de especificaciones de GCC para que apunte al nuevo enlazador dinámico.

```
SPECFILE=`gcc --print-file specs` &&
sed 's@ /lib/ld-linux.so.2@ /tools/lib/ld-linux.so.2@g' \
        $SPECFILE > tempspecfile &&
mv -f tempspecfile $SPECFILE &&
unset SPECFILE
```

6.5.6 GCC-3.4.3 - Fase 2

Ahora pueden reconstruirse GCC y Binutils enlazándolos con la nueva Glibc.

Se vuelve a crear un directorio de construcción dedicado:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

Se prepara GCC para su compilación:

```
../gcc-3.4.3/configure --prefix=/tools \
    --libexecdir=/tools/lib --with-local-prefix=/tools \
    --enable-clocale=gnu --enable-shared \
    --enable-threads=posix --enable-__cxa_atexit \
    --enable-languages=c,c++ --disable-libstdcxx-pch
```

Significado de las nuevas opciones de configure:

--enable-clocale=gnu

Esta opción asegura que se seleccione el modelo de locale correcto para las librerías de C++ en todos los casos. Si el guión configure encuentra instalada la locale *de_DE*, seleccionará el modelo correcto de *gnu*. Sin embargo, las personas que no instalan la locale *de_DE* pueden correr el riesgo de construir librerías de C++ incompatibles en la ABI debido a que se selecciona el modelo de locale genérico, que es incorrecto.

--enable-threads=posix

Esto activa el manejo de excepciones C++ para código multihilo.

--enable-__cxa_atexit

Esta opción permite el uso de *__cxa_atexit*, en vez de *atexit*, para registrar destructores C++ para objetos estáticos locales y objetos globales. Es esencial para un manejo de destructores completamente compatible con los estándares. También afecta al ABI de C++ obteniendo librerías compartidas y programas C++ interoperables con otras distribuciones Linux.

--enable-languages=c,c++

Esta opción asegura que se construyan tanto el compilador de C como el de C++.

--disable-libstdcxx-pch

No construye la cabecera precompilada (PCH) para libstdc++. Necesita mucho espacio

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.7 Binutils-2.15.94.0.2.2 - Fase 2

Se vuelve a crear un directorio dedicado para la construcción:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

Se prepara Binutils para su compilación:

```
../binutils-2.15.94.0.2.2/configure --prefix=/tools \
```

--disable-nls --enable-shared --with-lib-path=/tools/lib

Significado de la nueva opción de configure:

--with-lib-path=/tools/lib

Esto le indica al guión configure que especifique la ruta de búsqueda de librerías por defecto durante la compilación de Binutils, resultando en que se le pase /tools/lib al enlazador. Esto evita que el enlazador busque en los directorios de librerías del anfitrión.

Se compila el paquete:

make

Se instala el paquete:

make install

Se prepara el enlazador para la fase de “Reajuste” de la próxima sección:

make -C ld clean

make -C ld LIB_PATH=/usr/lib:/lib

6.5.8 Gawk-3.1.4

El paquete Gawk contiene programas para manipular ficheros de texto.

Se prepara Gawk para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.9 Coreutils-5.2.1

El paquete Coreutils contiene utilidades para mostrar y establecer las características básicas del sistema.

Se prepara Coreutils para su compilación:

DEFAULT_POSIX2_VERSION=199209 ./configure --prefix=/tools

Este paquete tiene un problema cuando se compila contra una versión de Glibc posterior a 2.3.2. Algunas de las utilidades de Coreutils (como **head**, **tail** y **sort**) rechazarán su sintaxis tradicional, la cual se ha usado desde hace aproximadamente unos 30 años. Esta vieja sintaxis está tan arraigada que debería preservarse la compatibilidad hasta que puedan actualizarse los múltiples sitios en la que se usa. La compatibilidad hacia atrás se consigue estableciendo en el anterior comando el valor de la variable de entorno DEFAULT_POSIX2_VERSION a “199209”.

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.10 Bzip2-1.0.3

El paquete Bzip2 contiene programas para comprimir y descomprimir ficheros.

El paquete Bzip2 no tiene un guión **configure** así que solo se compila:

make

Se instala el paquete:

make PREFIX=/tools install

6.5.11 Gzip-1.3.5

El paquete Gzip contiene programas para comprimir y descomprimir ficheros.

Se prepara Gzip para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.12 Diffutils-2.8.1

Contiene programas que muestran las diferencias entre ficheros o directorios.
Se prepara Diffutils para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.13 Findutils-4.2.23

El paquete Findutils contiene programas para encontrar ficheros.

Se suministran estos programas para hacer búsquedas recursivas en un árbol de directorios, y para crear, mantener y consultar una base de datos (más rápida que la búsqueda recursiva, pero imprecisa si la base de datos no se ha actualizado recientemente).

Se prepara Findutils para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.14 Make-3.80

El paquete Make contiene un programa para compilar paquetes.
Se prepara Make para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.15 Grep-2.5.1a

El paquete Grep contiene programas para buscar dentro de ficheros.

Se prepara Grep para su compilación:

**./configure --prefix=/tools **
--disable-perl-regexp

Significado de las opciones de configure:

--disable-perl-regexp

Esto asegura que **grep** no se enlaza contra alguna librería PCRE que pudiese estar presente en el anfitrión y que no estará disponible una vez que se entre en el entorno **chroot**.

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.16 Sed-4.1.4

El paquete Sed contiene un editor de flujos.

Se prepara Sed para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.17 Gettext-0.14.3

El paquete Gettext contiene utilidades para la internacionalización y localización. Esto permite a los programas compilarse con Soporte de Lenguaje Nativo (NLS), lo que les permite mostrar mensajes en el idioma nativo del usuario.

Se prepara Gettext para su compilación:

**./configure --prefix=/tools --disable-libasprintf **

--without-csharp

Significado de las opciones de configure:

--disable-libasprintf

Esta opción le indica a Gettext que no construya la librería asprintf. Puesto que nada en esta sección o la siguiente requiere dicha librería y Gettext se reconstruirá más adelante, se excluye para salvar tiempo y espacio.

--without-csharp

Esto le indica a Gettext que no construya el soporte para el compilador C#, que puede estar presente en el anfitrión pero no estará disponible cuando se entre en el entorno **chroot**.

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.18 Ncurses-5.4

El paquete Ncurses contiene librerías para el manejo de pantallas de caracteres independiente del terminal.

Se prepara Ncurses para su compilación:

**./configure --prefix=/tools --with-shared **
--without-debug --without-ada --enable-overwrite

Significado de las opciones de configure:

--without-ada

Esto asegura que Ncurses no construya su soporte para el compilador Ada, que puede estar presente en el anfitrión pero que no estará disponible al entrar en el entorno **chroot**.

--enable-overwrite

Esto le indica a Ncurses que instale sus ficheros de cabecera en /tools/include en vez de en /tools/include/.

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.19 Patch-2.5.4

El paquete Patch contiene un programa para modificar o crear ficheros mediante la aplicación de un fichero "parche" creado normalmente con el programa **diff**.

Se prepara Patch para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.20 Tar-1.15.1

El paquete Tar contiene un programa de archivado.

Se prepara Tar para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.21 Texinfo-4.8

El paquete Texinfo contiene programas usados para leer, escribir y convertir páginas inf

Se prepara Texinfo para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.22 Bash-3.0

El paquete Bash contiene la “Bourne-Again SHell”.

Bash tiene un problema cuando se compila contra las nuevas versiones de Glibc, causando una caída inapropiada. Este parche corrige el problema:

patch -Np1 -i ../bash-3.0-avoid_WCONTINUED-1.patch

Se prepara Bash para su compilación:

./configure --prefix=/tools --without-bash-malloc –-with-curses

Significado de la opción de configure:

--without-bash-malloc

Esta opción desactiva el uso de la función de ubicación de memoria (malloc) de Bash, que se sabe que provoca violaciones de segmento. Al desactivar esta opción Bash utilizará la función malloc de Glibc, que es más estable.

Se compila el paquete:

make

Se instala el paquete:

make install

Se crea un enlace para los programas que usan **sh** como intérprete de comandos:

ln -vs bash /tools/bin/sh

6.5.23 M4-1.4.3

El paquete M4 contiene un procesador de macros.

Se prepara M4 para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.24 Bison-2.0

El paquete Bison contiene un generador de analizadores sintácticos.

Se prepara Bison para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.25 Flex-2.5.31

El paquete Flex contiene una utilidad para generar programas que reconocen patrones de texto.

Flex contiene varios errores conocidos. Pueden corregirse con el siguiente parche:

patch -Np1 -i ../flex-2.5.31-debian_fixes-3.patch

Se prepara Flex para su compilación:

./configure --prefix=/tools

Se compila el paquete:

make

Se instala el paquete:

make install

6.5.26 Util-linux-2.12q

El paquete Util-linux contiene una miscelánea de utilidades. Entre otras hay utilidades para manejar sistemas de ficheros, consolas, particiones y mensajes.

Util-linux no utiliza las cabeceras y librerías recién instaladas en el directorio /tools. Esto se corrige modificando el guión configure:

sed -i 's@/usr/include@/tools/include@g' configure

Se prepara Util-linux para su compilación:

./configure

Se construyen algunas rutinas de soporte:

make -C lib

Sólo es necesario construir algunas de las utilidades incluidas en este paquete:

make -C mount mount umount

make -C text-utils more

Se copia estos programas al directorio de herramientas temporales:

cp mount/{,u}mount text-utils/more /tools/bin

6.5.27 Perl-5.8.7

El paquete Perl contiene el Lenguaje Práctico de Extracción e Informe.

Se aplica el siguiente parche para corregir algunas rutas a la librería C fijadas en el código:

patch -Np1 -i ../perl-5.8.7-libc-1.patch

Se prepara Perl para su compilación:

./configure.gnu --prefix=/tools -Dstatic_ext='IO Fcntl POSIX'

Significado de la opción de configure:

-Dstatic_ext='IO Fcntl POSIX'

Esto le indica a Perl que construya el conjunto mínimo de extensiones estáticas necesarias para ejecutar el banco de pruebas de Coreutils en la siguiente sección.

Sólo es necesario construir algunas de las utilidades incluidas en este paquete:

make perl utilities

Se Instala estas herramientas y sus librerías:

cp -v perl pod/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.8.7
cp -Rv lib/* /tools/lib/perl5/5.8.7

6.5.28 Eliminación de Símbolos

Los binarios y librerías que se han construido contienen unos 130 MB de símbolos de depuración innecesarios. Se elimina esos símbolos con:

strip --strip-debug /tools/lib/*
strip --strip-unneeded /tools/{,s}bin/*

Para recuperar otros 30 MB, se elimina la documentación:

rm -rf /tools/{info,man}

6.6 Sección 5

En esta sección se entra en la zona de edificación y se comienza a construir de verdad el nuevo sistema. Es decir, se cambia la raíz al mini sistema Linux temporal construido, se hacen unos cuantos preparativos finales, y entonces se comienza a instalar los paquetes.

6.6.1 Montar los sistemas de ficheros virtuales del núcleo

Varios sistemas de ficheros exportados por el núcleo son usados para comunicarse hacia y desde el propio núcleo. Estos sistemas de ficheros son virtuales y no utilizan espacio en disco. El contenido del sistema de ficheros reside en memoria.

Se comienza creando los directorios sobre los que se montarán dichos sistemas de ficheros:

mkdir -pv $LFS/{proc,sys}

Ahora se monta los sistemas de ficheros:

mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys

Pronto se montarán sistemas de ficheros adicionales desde dentro del entorno chroot. Para mantener el anfitrión actualizado, se realiza ahora un “falso montaje” para cada uno de ellos:

mount -vft tmpfs tmpfs $LFS/dev
mount -vft tmpfs tmpfs $LFS/dev/shm
mount -vft devpts -o gid=4,mode=620 devpts $LFS/dev/pts

6.6.2 Entrar al entorno chroot

Es hora de entrar en el entorno chroot para iniciar la construcción e instalar el sistema final. Como usuario *root*, se ejecuta el siguiente comando para entrar a un mundo que está, en este momento, poblado sólo por las herramientas temporales.

```
chroot "$LFS" /tools/bin/env -i \
        HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
        PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \
        /tools/bin/bash --login +h
```

La opción *-i* pasada al comando **env** limpiará todas las variables del chroot. Después de esto solamente se establecen de nuevo las variables HOME, TERM, PS1 y PATH. La construcción *TERM=$TERM* establece la variable TERM dentro del chroot al mismo valor que tiene fuera del chroot. Dicha variable es necesaria para que funcionen correctamente programas como **vim** y **less**.

6.6.3 Creación de los directorios

Es hora de crear cierta estructura en el sistema de ficheros. Se crea un árbol de directorios estándar ejecutando los siguientes comandos:

```
install -dv /{bin,boot,dev,etc/opt,home,lib,mnt}
install -dv /{sbin,srv,usr/local,var,opt}
install -dv /root -m 0750
install -dv /tmp /var/tmp -m 1777
install -dv /media/{floppy,cdrom}
install -dv /usr/{bin,include,lib,sbin,share,src}
ln -sv share/{man,doc,info} /usr
install -dv /usr/share/{doc,info,locale,man}
install -dv /usr/share/{misc,terminfo,zoneinfo}
install -dv /usr/share/man/man{1,2,3,4,5,6,7,8}
install -dv /usr/local/{bin,etc,include,lib,sbin,share,src}
ln -sv share/{man,doc,info} /usr/local
install -dv /usr/local/share/{doc,info,locale,man}
install -dv /usr/local/share/{misc,terminfo,zoneinfo}
install -dv /usr/local/share/man/man{1,2,3,4,5,6,7,8}
install -dv /var/{lock,log,mail,run,spool}
install -dv /var/{opt,cache,lib/{misc,locate},local}
install -dv /opt/{bin,doc,include,info}
install -dv /opt/{lib,man/man{1,2,3,4,5,6,7,8}}
```

Los directorios se crean por defecto con los permisos 755, pero esto no es deseable para todos los directorios. En los comandos anteriores se hacen dos cambios: uno para el directorio personal de *root* y otro para los directorios de los ficheros temporales.

El primer cambio asegura que nadie aparte de root pueda entrar en el directorio /root. El segundo cambio asegura que cualquier usuario pueda escribir en los directorios /tmp y /var/tmp, pero no pueda borrar los ficheros de otros usuarios. Esto último lo prohíbe el llamado “bit pegajoso” (sticky bit), el bit más alto (1) en la máscara de permisos 1777.

El árbol de directorios está basado en el estándar FHS.

6.6.4 Creación de enlaces simbólicos esenciales

Algunos programas tienen fijadas en su código rutas a programas que aún no existen. Para satisfacer a estos programas se crean unos cuantos enlaces simbólicos que serán sustituidos por ficheros reales durante el transcurso de esta sección a medida que se vayan instalando todos los programas.

```
ln -sv /tools/bin/{bash,cat,pwd,stty} /bin
ln -sv /tools/bin/perl /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1} /usr/lib
ln -sv bash /bin/sh
```

6.6.5 Creación de los ficheros de contraseñas, grupos y registro

Para que *root* pueda entrar al sistema y para que el nombre “root” sea reconocido, es necesario tener las entradas apropiadas en los ficheros /etc/passwd y /etc/group.

Se crea el fichero /etc/passwd ejecutando el siguiente comando:

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
EOF
```

La contraseña real para *root* (la “x” es sólo un sustituto) se establecerá más adelante.

Se crea el fichero /etc/group ejecutando el siguiente comando:

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
```

tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
EOF

Los grupos creados no son parte de ningún estándar, son grupos escogidos en parte por los requisitos de la configuración de Udev en esta sección, y en parte por la práctica común empleada por una serie de distribuciones Linux existentes.

Para eliminar el "I have no name!" del símbolo del sistema, se inicia un nuevo intérprete de comandos. Puesto que se instalo una Glibc completa en la sección 4 y se acaba de crear los ficheros /etc/passwd y /etc/group, la resolución de nombres de usuarios y grupos funcionará ahora.

exec /tools/bin/bash --login +h

Los programas **login**, **getty** e **init** (entre otros) mantienen una serie de ficheros de registro con información sobre quienes están y estaban dentro del sistema. Sin embargo, estos programas no crean dichos ficheros si no existen. Se crea los ficheros de registro con sus permisos correctos:

touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}
chgrp -v utmp /var/run/utmp /var/log/lastlog
chmod -v 664 /var/run/utmp /var/log/lastlog

El fichero /var/run/utmp lista los usuarios que están actualmente dentro del sistema, /var/log/wtmp registra todos los ingresos y salidas. El fichero /var/log/lastlog muestra, para cada usuario, cuando fue la última vez que ingresó, y el fichero /var/log/btmp lista los intentos de ingreso fallidos.

6.6.6 Poblar /dev

Cuando el núcleo arranca el sistema necesita la presencia de ciertos nodos de dispositivo, en concreto los dispositivos console y null. Los dispositivos se crearán en el disco duro para que estén disponibles antes de que **udev** sea iniciado y, adicionalmente, cuando Linux es iniciado en modo de usuario único (de aquí los permisos restrictivos en console). Se crea los dispositivos ejecutando los siguientes comandos:

```
mknod -m 600 /dev/console c 5 1
mknod -m 666 /dev/null c 1 3
```

El método recomendado para poblar /dev con dispositivos es montar un sistema de ficheros virtual (como tmpfs) en /dev, y permitir que los dispositivos se creen dinámicamente en el sistema de ficheros virtual a medida que son detectados o accedidos. Esto se hace generalmente durante el proceso de arranque. Puesto que este nuevo sistema no ha sido arrancado, es necesario hacer lo que hubiesen hecho los guiones de arranque y montar /dev:

```
mount -nvt tmpfs none /dev
```

El paquete Udev es quien crea en realidad los dispositivos en el directorio /dev. Puesto que no se instalará hasta más adelante, se crea manualmente el conjunto mínimo de nodos de dispositivo necesarios para completar la construcción de este sistema:

```
mknod -m 622 /dev/console c 5 1
mknod -m 666 /dev/null c 1 3
mknod -m 666 /dev/zero c 1 5
mknod -m 666 /dev/ptmx c 5 2
mknod -m 666 /dev/tty c 5 0
```

mknod -m 444 /dev/random c 1 8
mknod -m 444 /dev/urandom c 1 9
chown -v root:tty /dev/{console,ptmx,tty}

Ciertos enlaces simbólicos y directorios requeridos por el sistema son creados durante el arranque por el paquete LFS-Bootscripts. Puesto que este es un entorno **chroot** y no un entorno iniciado, se necesita crear aquí dichos enlaces y directorios:

ln -sv /proc/self/fd /dev/fd
ln -sv /proc/self/fd/0 /dev/stdin
ln -sv /proc/self/fd/1 /dev/stdout
ln -sv /proc/self/fd/2 /dev/stderr
ln -sv /proc/kcore /dev/core
mkdir -v /dev/pts
mkdir -v /dev/shm

Finalmente, se monta los sistemas de ficheros virtuales (del núcleo) adecuados en los directorios recién creados:

mount -vt devpts -o gid=4,mode=620 none /dev/pts
mount -vt tmpfs none /dev/shm

6.6.7 Linux-Libc-Headers-2.6.11.2

El paquete Linux-Libc-Headers contiene las cabeceras “saneadas” del núcleo.

Se Instala los ficheros de cabecera:

cp -Rv include/asm-i386 /usr/include/asm
cp -Rv include/linux /usr/include

Se asegura de que todas las cabeceras son propiedad de root:

chown -Rv root:root /usr/include/{asm,linux}

Se asegura de que los usuarios pueden leer las cabeceras:

find /usr/include/{asm,linux} -type d -exec chmod -v 755 {} \;
find /usr/include/{asm,linux} -type f -exec chmod -v 644 {} \;

6.6.8 Glibc-2.3.4

El sistema de construcción de Glibc está muy bien auto contenido y se instalará perfectamente, incluso aunque el fichero de especificaciones del compilador y los guiones del enlazador todavía apunten a /tools. No se puede ajustar las especificaciones y el enlazador antes de instalar Glibc, porque entonces las comprobaciones del autoconf de Glibc darían resultados erróneos y esto arruinaría el objetivo de conseguir una construcción limpia.

El paquete linuxthreads contiene las páginas de manual para las librerías de hilos instaladas por Glibc. Se descomprime dentro del directorio de las fuentes de Glibc:

tar -xjvf ../glibc-linuxthreads-2.3.4.tar.bz2

En ciertas circunstancias raras, Glibc puede fallar cuando los directorios de búsqueda estándar no existen. El siguiente parche evita esto:

patch -Np1 -i ../glibc-2.3.4-rtld_search_dirs-1.patch

Hay dos pruebas de Glibc que fallan cuando el núcleo en ejecución es un 2.6.11.x. Se ha determinado que el problema se encuentra en las propias pruebas, no en libc o en el núcleo. Este parche corrige el problema:

patch -Np1 -i ../glibc-2.3.4-fix_test-1.patch

Se aplica el siguiente parche para corregir un error en Glibc que impide la ejecución de ciertos programas (como OpenOffice.org):

patch -Np1 -i ../glibc-2.3.4-tls_assert-1.patch

La documentación de Glibc recomienda construirlo fuera del árbol de las fuentes, en un directorio de construcción dedicado:

mkdir -v ../glibc-build
cd ../glibc-build

Se prepara Glibc para su compilación:

**../glibc-2.3.4/configure --prefix=/usr **
**--disable-profile --enable-add-ons **
--enable-kernel=2.6.0 --libexecdir=/usr/lib/glibc

Significado de la nueva opción de configure:

--libexecdir=/usr/lib/glibc
Esto cambia la localización del programa **pt_chown** de su ubicación por defecto /usr/libexec a /usr/lib/glibc.

Se compila el paquete:

make

Se instala el paquete:

make install

Las locales que hacen que tu sistema responda en un idioma diferente no se instalaron con el comando anterior. Hazlo con este:
make localedata/install-locales

Se construye las páginas de manual de linuxthreads, que son una gran referencia sobre la API de hilos (aplicable también a NPTL):

make -C ../glibc-2.3.4/linuxthreads/man

Se instala dichas páginas:

make -C ../glibc-2.3.4/linuxthreads/man install

Se necesita crear el fichero /etc/nsswitch.conf, porque aunque Glibc facilita los valores por defecto cuando este fichero no se encuentra o está corrupto, estos valores por defecto no funcionan bien en un entorno de red. También hay que configurar la zona horaria.

Se crea un nuevo fichero /etc/nsswitch.conf ejecutando lo siguiente:

cat > /etc/nsswitch.conf << "EOF"
Inicio de /etc/nsswitch.conf

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

protocols: files
services: files
ethers: files
rpc: files

Fin de /etc/nsswitch.conf
EOF

Para determinar la zona horaria local, se ejecuta el siguiente guión:

tzselect

Se crea el fichero /etc/localtime ejecutando:

**cp -v --remove-destination /usr/share/zoneinfo/*America/Caracas* **
/etc/localtime

Significado de la opción de cp:

--remove-destination

Esto es necesario para forzar la eliminación del enlace simbólico que ya existe. La razón por la que copiamos en lugar de enlazar es para cubrir el caso en el que /usr está en otra partición. Esto puede ser importante cuando se arranca en modo de usuario único.

6.6.9 Reajustar las herramientas

Ahora que se ha instalado las librerías de C finales, es hora de ajustar de nuevo el conjunto de herramientas. Se ajustan para que enlacen cualquier nuevo programa compilado contra estas nuevas librerías.
Se comienza ajustando el enlazador. Para ello se conserva los directorios de fuentes y de construcción de la segunda fase de Binutils. Se instala el enlazador ajustado ejecutando el siguiente comando desde el directorio binutils-build:

make -C ld INSTALL=/tools/bin/install install

Desde ahora todos los programas que se compilen se enlazarán solamente contra las librerías que hay en /usr/lib y /lib.

Se elimina los directorios de fuentes y de construcción de Binutils.

Se corrige el fichero specs de GCC para que apunte al nuevo enlazador dinámico.

```
perl -pi -e 's@ /tools/lib/ld-linux.so.2@ /lib/ld-linux.so.2@g;' \
        -e 's@\*startfile_prefix_spec:\n@$_/usr/lib/ @g;' \
        `gcc --print-file specs`
```

6.6.10 Binutils-2.15.94.0.2.2

La documentación de Binutils recomienda construirlo fuera del árbol de las fuentes, en un directorio de construcción dedicado:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

Se prepara Binutils para su compilación:

```
../binutils-2.15.94.0.2.2/configure --prefix=/usr \
        --enable-shared
```

Se compila el paquete:

```
make tooldir=/usr
```

Se instala el paquete:

```
make tooldir=/usr install
```

Se instala el fichero de cabecera libiberty, pues lo necesitan algunos paquetes:

```
cp -v ../binutils-2.15.94.0.2.2/include/libiberty.h /usr/incluye
```

6.6.11 GCC-3.4.3

Se aplica sólo el parche No-Fixincludes:

patch -Np1 -i ../gcc-3.4.3-no_fixincludes-1.patch

GCC falla al compilar algunos paquetes cuando se usa en conjunción con la más nueva versión de Binutils. Se aplica el siguiente parche para corregir dicho problema:

patch -Np1 -i ../gcc-3.4.3-linkonce-1.patch

Se aplica una sustitución **sed** que suprimirá la instalación de libiberty.a. Se usará en su lugar la versión de libiberty.a suministrada por Binutils:

sed -i 's/install_to_$(INSTALL_DEST) //' libiberty/Makefile.in

La documentación de GCC recomienda construirlo fuera del árbol de las fuentes, en un directorio de construcción dedicado:

mkdir -v ../gcc-build
cd ../gcc-build

Se prepara GCC para su compilación:

```
../gcc-3.4.3/configure --prefix=/usr \
      --libexecdir=/usr/lib --enable-shared \
      --enable-threads=posix --enable-__cxa_atexit \
      --enable-clocale=gnu --enable-languages=c,c++
```

Se compila el paquete:

make

Se instala el paquete:

make install

Algunos paquetes esperan que el preprocesador de C esté instalado en el directorio /lib. Para dar soporte a estos paquetes, se crea un enlace simbólico:

ln -sv ../usr/bin/cpp /lib

Muchos paquetes usan el nombre **cc** para llamar al compilador de C. Para satisfacer a estos paquetes, se crea un enlace simbólico:

ln -sv gcc /usr/bin/cc

6.6.12 Coreutils-5.2.1

Un problema conocido en el programa **uname** de este paquete es que la opción *-p* siempre devuelve unknown (desconocido). El siguiente parche corrige este comportamiento en arquitecturas Intel:

patch -Np1 -i ../coreutils-5.2.1-uname-2.patch

Se evita que Coreutils instale binarios que serán instalados más tarde por otros paquetes:

patch -Np1 -i ../coreutils-5.2.1-suppress_uptime_kill_su-1.patch

Se prepara Coreutils para su compilación:

DEFAULT_POSIX2_VERSION=199209 ./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

Se mueve los programas a la localización especificada por el FHS:

mv -v /usr/bin/{cat,chgrp,chmod,chown,cp,date,dd,df,echo} /bin
mv -v /usr/bin/{false,hostname,ln,ls,mkdir,mknod,mv,pwd,rm} /bin
mv -v /usr/bin/{rmdir,stty,sync,true,uname} /bin
mv -v /usr/bin/chroot /usr/sbin

Algunos de los guiones del paquete LFS-Bootscripts dependen de **head** y **sleep**. Como /usr puede no estar disponible en las primeras fases del arranque, es necesario que estos binarios se encuentren en la partición raíz:

mv -v /usr/bin/{head,sleep} /bin

6.6.13 Zlib-1.2.3

El paquete Zlib contiene rutinas de compresión y descompresión usadas por algunos programas.

Se prepara Zlib para su compilación:

./configure --prefix=/usr --shared --libdir=/lib

Se compila el paquete:

make

Se instala la librería compartida:

make install

El comando anterior instalo un fichero .so en /lib. Se elimina y se reenlaza a /usr/lib:

rm -v /lib/libz.so
ln -sfv ../../lib/libz.so.1.2.3 /usr/lib/libz.so

Se construye también la librería estática:

./configure --prefix=/usr
make

Se instala la librería estática:

make install

Se corrige los permisos de la librería estática:

chmod -v 644 /usr/lib/libz.a

6.6.14 Iana-Etc-1.04

El paquete Iana-Etc contiene datos de servicios y protocolos de red.

El siguiente comando convierte los datos crudos proporcionados por IANA a formatos correctos para los ficheros de datos /etc/protocols y /etc/services:

make

Se instala el paquete:

make install

6.6.15 Findutils-4.2.23

Se prepara Findutils para su compilación:

**./configure --prefix=/usr --libexecdir=/usr/lib/locate **
--localstatedir=/var/lib/locate

Significado de la opción de configure:

--localstatedir

Esta opción cambia la localización de la base de datos de **locate** para que se encuentre en /var/lib/locate, que cumple el FHS.

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.16 Gawk-3.1.4

Se prepara Gawk para su compilación:

./configure --prefix=/usr --libexecdir=/usr/lib

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.17 Ncurses-5.4

Se prepara Ncurses para su compilación:

./configure --prefix=/usr --with-shared --without-debug

Se compila el paquete:

make

Se instala el paquete:

make install

Se otorga permisos de ejecución a las librerías Ncurses:

chmod -v 755 /usr/lib/*.5.4

Se corrige una librería que no debería ser ejecutable:

chmod -v 644 /usr/lib/libncurses++.a

Se mueve las librerías al directorio /lib, que es donde se espera que residan:

mv -v /usr/lib/libncurses.so.5* /lib

Debido a que se han movido las librerías, algunos enlaces simbólicos apuntan a ficheros que no existen. Se regenera esos enlaces simbólicos:

ln -sfv ../../lib/libncurses.so.5 /usr/lib/libncurses.so

ln -sfv libncurses.so /usr/lib/libcurses.so

6.6.18 Readline-5.0

El paquete Readline contiene un conjunto de librerías que ofrecen edición de la línea de comandos y capacidades de historial.

El siguiente parche incluye una corrección de un problema por el que Readline, en ocasiones, muestra sólo 33 caracteres en una línea y salta a la siguiente línea. También incluye otras correcciones recomendadas por el autor de Readline:

patch -Np1 -i ../readline-5.0-fixes-1.patch

Se prepara Readline para su compilación:

./configure --prefix=/usr --libdir=/lib

Se compila el paquete:

make SHLIB_XLDFLAGS=-lncurses

Significado de la opción de make:

SHLIB_XLDFLAGS=-lncurses

Esta opción fuerza a Readline a enlazarse contra la librería libncurses.

Se instala el paquete:

make install

Se asigna a las librerías dinámicas de Readline unos permisos más apropiados:

chmod -v 755 /lib/lib{readline,history}.so*

Se mueve las librerías estáticas a una ubicación más correcta:

mv -v /lib/lib{readline,history}.a /usr/lib

Ahora se elimina los ficheros .so del directorio /lib y se reenlaza a /usr/lib:

rm -v /lib/lib{readline,history}.so
ln -sfv ../../lib/libreadline.so.5 /usr/lib/libreadline.so
ln -sfv ../../lib/libhistory.so.5 /usr/lib/libhistory.so

6.6.19 M4-1.4.3

Se prepara M4 para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.20 Bison-2.0

Se prepara Bison para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.21 Less-382

El paquete Less contiene un visor de ficheros de texto.

Se prepara Less para su compilación:

./configure --prefix=/usr --bindir=/bin --sysconfdir=/etc

Significado de la opción de configure:

--sysconfdir=/etc

Esta opción le indica al programa creado por el paquete que busque en /etc sus ficheros de configuración.

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.22 Sed-4.1.4

Por defecto Sed instala su documentación HTML en /usr/share/doc. Se cambia esto a /usr/share/doc/sed-4.1.4 aplicando el siguiente comando **sed**:

sed -i 's@/doc@&/sed-4.1.4@' doc/Makefile.in

Se prepara Sed para su compilación:

./configure --prefix=/usr --bindir=/bin

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.23 Flex-2.5.31

Flex contiene varios errores conocidos. Se corrigen con el siguiente parche:

patch -Np1 -i ../flex-2.5.31-debian_fixes-3.patch

Se prepara Flex para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

Ciertos paquetes esperan encontrar la librería lex en el directorio /usr/lib. Se crea un enlace simbólico para solventar esto:

ln -sv libfl.a /usr/lib/libl.a

Algunos programas aún no conocen **flex** e intentan encontrar a su predecesor **lex**. Para complacer a estos programas, se crea un guión envoltorio de nombre lex que llame a **flex** en modo de emulación lex:

cat > /usr/bin/lex << "EOF"

```
#!/bin/sh
# Inicio de /usr/bin/lex

exec /usr/bin/flex -l "$@"

# Fin de /usr/bin/lex
```
EOF

chmod -v 755 /usr/bin/lex

6.6.24 Gettext-0.14.3

Se prepara Gettext para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.25 Inetutils-1.4.2

El paquete Inetutils contiene programas para trabajo básico en red.
Inetutils tiene ciertos problemas con los núcleos Linux de la serie 2.6. Se corrige aplicando el siguiente parche:

patch -Np1 -i ../inetutils-1.4.2-kernel_headers-1.patch

Se prepara Inetutils para su compilación:

```
./configure --prefix=/usr --libexecdir=/usr/sbin \
        --sysconfdir=/etc --localstatedir=/var \
        --disable-logger --disable-syslogd \
        --disable-whois --disable-servers
```

Significado de las opciones de configure:

--disable-logger

Esta opción evita que Inetutils instale el programa **logger**, que sirve para que los guiones le pasen mensajes al Demonio de Registro de Eventos del Sistema.

--disable-syslogd

Esta opción evita que Inetutils instale el Demonio de Registro de Eventos del Sistema, que será instalado con el paquete Sysklogd.

--disable-whois

Esta opción desactiva la construcción del cliente **whois** de Inetutils, que está demasiado anticuado.

--disable-servers

Esto desactiva la construcción de los diferentes servidores incluidos como parte del paquete Inetutils.

Se compila el paquete:

make

Se instala el paquete:

make install

Se mueve el programa **ping** al lugar indicado por el FHS:

mv -v /usr/bin/ping /bin

6.6.26 IPRoute2-2.6.11-050330

El paquete IPRoute2 contiene programas para el trabajo básico y avanzado en redes basadas en IPV4.

Se prepara IPRoute2 para su compilación:

./configure

Se compila el paquete:

make SBINDIR=/sbin

Significado de la opción de make:

SBINDIR=/sbin

Esto asegura que los binarios de IPRoute2 se instalarán en /sbin. Esta es la localización correcta según el FHS, pues algunos de los binarios de IPRoute2 se utilizan en los guiones de arranque.

Se instala el paquete:

make SBINDIR=/sbin install

6.6.27 Perl-5.8.7

Se prepara Perl para su compilación con:

./configure.gnu --prefix=/usr -Dpager="/bin/less -isR"

Significado de la opción de configure:

-Dpager="/bin/less -isR"

Esto corrige un error en el modo en que **perldoc** invoca al programa **less**.

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.28 Texinfo-4.8

Texinfo permite a usuarios locales sobrescribir ficheros arbitrarios mediante un ataque de enlace simbólico sobre ficheros temporales. Se aplica el siguiente parche para corregir esto:

patch -Np1 -i ../texinfo-4.8-tempfile_fix-1.patch

Se prepara Texinfo para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.29 Autoconf-2.59

El paquete Autoconf contiene programas para generar guiones del intérprete de comandos que pueden configurar automáticamente el código fuente.

Se prepara Autoconf para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.30 Automake-1.9.5

El paquete Automake contiene programas para generar Makefiles que se utilizan con Autoconf.

Se prepara Automake para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.31 Bash-3.0

El siguiente parche corrige varios problemas, incluido uno por el que Bash en ocasiones sólo mostrará 33 caracteres en una línea y saltará a la siguiente:

patch -Np1 -i ../bash-3.0-fixes-3.patch

Bash también tiene problemas cuando se compila contra las nuevas versiones de Glibc. El siguiente parche resuelve este problema:

patch -Np1 -i ../bash-3.0-avoid_WCONTINUED-1.patch

Se prepara Bash para su compilación:

**./configure --prefix=/usr --bindir=/bin **
--without-bash-malloc --with-installed-readline

Significado de la opción de configure:

--with-installed-readline

Esta opción le indica a Bash que utilice la librería readline que se encuentra en el sistema, en vez de utilizar su propia versión de Readline.

Se compila el paquete:

make

Se instala el paquete:

make install

Se lanza el programa **bash** recién compilado

exec /bin/bash --login +h

6.6.32 Libtool-1.5.14

El paquete Libtool contiene el guión de GNU para soporte genérico de librerías. Oculta la complejidad del uso de librerías compartidas tras una interfaz consistente y portable.

Se prepara Libtool para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.33 Bzip2-1.0.3

Se prepara Bzip2 para su compilación:

make -f Makefile-libbz2_so
make clean

La opción *-f* provocará que Bzip2 sea construido usando un fichero Makefile diferente, en este caso el fichero Makefile-libbz2_so, el cual crea una librería dinámica libbz2.so y enlaza las utilidades de Bzip2 con ella.

Se compila el paquete:

make

Se instala los programas:

make install

Se instala el binario dinámico **bzip2** en el directorio /bin, se crea algunos enlaces simbólicos necesarios y se hace limpieza:

cp -v bzip2-shared /bin/bzip2
cp -av libbz2.so* /lib
ln -sv ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm -v /usr/bin/{bunzip2,bzcat,bzip2}
ln -sv bzip2 /bin/bunzip2
ln -sv bzip2 /bin/bzcat

6.6.34 Diffutils-2.8.1

Se prepara Diffutils para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.35 E2fsprogs-1.37

El paquete E2fsprogs contiene las utilidades para manejar el sistema de ficheros ext2. También soporta los sistemas de ficheros ext3 con registro de transacciones.

Se recomienda construir E2fsprogs en un subdirectorio del árbol de las fuentes:

```
mkdir -v build
cd build
```

Se prepara E2fsprogs para su compilación:

```
../configure --prefix=/usr --with-root-prefix="" \
      --enable-elf-shlibs --disable-evms
```

Significado de las opciones de configure:

--with-root-prefix=""

Ciertos programas (como el programa **e2fsck**) se consideran esenciales. Cuando, por ejemplo, /usr no está montado, estos programas esenciales deben estar disponibles. Pertenecen a directorios como /lib y /sbin.

--enable-elf-shlibs

Esto crea las librerías compartidas utilizadas por algunos de los programas de este paquete.

--disable-evms

Esto desactiva la construcción del módulo para el Enterprise Volume Management System (EVMS, Sistema Empresarial de Manejo de Volúmenes).

Se compila el paquete:

make

Se instala los binarios y la documentación:

make install

Se instala las librerías compartidas:

make install-libs

6.6.36 Grep-2.5.1a

Se prepara Grep para su compilación:

./configure --prefix=/usr --bindir=/bin

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.37 GRUB-0.96

El paquete GRUB contiene el GRand Unified Bootloader (Gran Gestor de Arranque Unificado).

Se prepara GRUB para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install
mkdir -v /boot/grub
cp -v /usr/lib/grub/i386-pc/stage{1,2} /boot/grub

6.6.38 Hotplug-2004_09_23

El paquete Hotplug contiene guiones que reaccionan a eventos hotplug (conexión de dispositivos en caliente) generados por el núcleo. Dichos eventos corresponden a cada cambio en el estado del núcleo visible en el sistema de ficheros sysfs, por ejemplo, la adición o eliminación de hardware. Este paquete detecta también el hardware existente durante el arranque e inserta los módulos necesarios dentro del núcleo en ejecución.

Se instala el paquete:

make install

cp -v etc/hotplug/pnp.distmap /etc/hotplug

Se elimina el guión de inicio instalado por Hotplug

rm -rfv /etc/init.d

La conexión en caliente de dispositivos de red no está aún soportada por los guiones de arranque del sistema. Por este motivo, se elimina el agente Hotplug de red:

rm -fv /etc/hotplug/net.agent

Se crea un directorio para almacenar el firmware que puede ser cargado por **hotplug**:

mkdir -v /lib/firmware

6.6.39 Make-3.80

Se prepara Make para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.40 Module-Init-Tools-3.1

El paquete Module-Init-Tools contiene programas para manejar módulos del núcleo en núcleos Linux con versión mayor o igual a 2.5.47.

Module-Init-Tools intenta reescribir su página de manual modprobe.conf durante el proceso de construcción. Esto es innecesario y además depende de **docbook2man**, que no se instala en el sistema. Se ejecuta el siguiente comando para evitar esto:

touch modprobe.conf.5

Se prepara Module-Init-Tools para su compilación:

./configure --prefix="" --enable-zlib

Significado de la opción de configure:

--enable-zlib

Esto permite al paquete Module-Init-Tools manejar módulos del núcleo comprimidos.

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.41 Patch-2.5.4

Se prepara Patch para su compilación:

./configure --prefix=/usr

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.42 Procps-3.2.5

El paquete Procps contiene programas para monitorizar procesos.

Se compila el paquete:

make

Se instala el paquete:

make install

6.6.43 Shadow-4.0.9

El paquete Shadow contiene programas para manejar contraseñas de forma segura.

Se prepara Shadow para su compilación:

./configure --libdir=/lib --enable-shared

Se suprime la instalación del programa **groups** y su página de manual, pues Coreutils proporciona una versión mejor:

sed -i 's/groups$(EXEEXT) //' src/Makefile
sed -i '/groups/d' man/Makefile

Se compila el paquete:

make

Se instala el paquete:

make install

Shadow utiliza dos ficheros para configurar los ajustes de autenticación para el sistema. Se instala estos ficheros de configuración:

cp -v etc/{limits,login.access} /etc

En vez de usar el método por defecto, *crypt*, se utiliza el método de encriptación de contraseñas *MD5*, que es más seguro y además permite contraseñas de más de 8 caracteres. También es necesario cambiar la obsoleta localización /var/spool/mail, que Shadow utiliza por defecto para los buzones de los usuarios, a /var/mail, que es la localización usada hoy en día. Ambas cosas pueden hacerse modificando el fichero de configuración correspondiente mientras se copia a su destino:

```
sed -e's@#MD5_CRYPT_ENAB.no@MD5_CRYPT_ENAB yes@' \
    -e 's@/var/spool/mail@/var/mail@' \
    etc/login.defs.linux > /etc/login.defs
```

Se mueve un programa mal ubicado a su lugar correcto:

mv -v /usr/bin/passwd /bin

Se mueve las librerías de Shadow a un lugar más apropiado:

mv -v /lib/libshadow.*a /usr/lib
rm -v /lib/libshadow.so
ln -sfv ../../lib/libshadow.so.0 /usr/lib/libshadow.so

La opción *-D* del programa **useradd** requiere el directorio /etc/default para funcionar correctamente:

mkdir -v /etc/default

Este paquete contiene utilidades para añadir, modificar o eliminar usuarios y grupos, establecer y cambiar sus contraseñas y otras tareas administrativas.

Para habilitar las contraseñas ocultas, se ejecuta el siguiente comando:

pwconv

Para habilitar las contraseñas de grupo ocultas, se ejecuta:

grpconv

Se elige una contraseña para el usuario *root* y se estable mediante:

passwd root

6.6.44 Sysklogd-1.4.1

El paquete Sysklogd contiene programas para registrar los mensajes del sistema, como aquellos generados por el núcleo cuando sucede algo inusual.

El siguiente parche corrige varios problemas, incluido un problema cuando se construye Sysklogd con los núcleos de las series Linux 2.6.

patch -Np1 -i ../sysklogd-1.4.1-fixes-1.patch

Se compila el paquete:

make

Se instala el paquete:

make install

Se crea un nuevo fichero /etc/syslog.conf ejecutando lo siguiente:

cat > /etc/syslog.conf << "EOF"

```
# Inicio de /etc/syslog.conf

auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *

# registra la salida de los guiones de arranque:
local2.* -/var/log/boot.log

# Fin de /etc/syslog.conf
```

EOF

6.6.45 Sysvinit-2.86

El paquete Sysvinit contiene programas para controlar el arranque, ejecución y cierre del sistema.

Cuando se cambia de nivel de ejecución (por ejemplo cuando apagamos el sistema) el programa **init** envía las señales de finalización a aquellos procesos que él mismo inició y que no deben estar en ejecución en el nuevo nivel. Mientras lo hace, **init** muestra mensajes del tipo “Sending processes the TERM signal” (Enviando la señal TERM a los procesos), que parece indicar que se está enviando dicha señal a todos los procesos que hay en ejecución. Para evitar esta confusión, se puede modificar las fuentes para que ese mensaje diga en su lugar “Sending processes started by init the TERM signal” (Enviando la señal TERM a los procesos iniciados por init):

```
sed -i 's@Sending processes@& started by init@g' \
        src/init.c
```

Se compila el paquete:

```
make -C src
```

Se instala el paquete:

```
make -C src install
```

Se crea un nuevo fichero /etc/inittab ejecutando lo siguiente:

```
cat > /etc/inittab << "EOF"
# Inicio de /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit
```

```
l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty -I '\033(K' tty1 9600
2:2345:respawn:/sbin/agetty -I '\033(K' tty2 9600
3:2345:respawn:/sbin/agetty -I '\033(K' tty3 9600
4:2345:respawn:/sbin/agetty -I '\033(K' tty4 9600
5:2345:respawn:/sbin/agetty -I '\033(K' tty5 9600
6:2345:respawn:/sbin/agetty -I '\033(K' tty6 9600

# Fin de /etc/inittab
```
EOF

La opción *-I '\033(K'* le indica a **agetty** que envíe al terminal esta secuencia de escape antes de hacer nada más.

6.6.46 Udev-056

El paquete Udev contiene programas para la creación dinámica de nodos de dispositivos. Se compila el paquete:

make udevdir=/dev

Significado de la opción de make:

udevdir=/dev

Esto le indica a **udev** en qué directorio se deben crear los nodos de dispositivos.

Se instala el paquete:

make DESTDIR=/ udevdir=/dev install

Significado de la opción de make:

DESTDIR=/

Esto evita que el proceso de instalación de Udev mate cualquier proceso **udevd** que pueda estar ejecutándose en el sistema anfitrión.

La configuración por defecto de Udev no es la ideal, así que se instala aquí los ficheros de configuración:

cp -v ../udev-config-4.rules /etc/udev/rules.d/25-lfs.rules

Se ejecuta el programa **udevstart** para crear el conjunto completo de nodos de dispositivos.

/sbin/udevstart

6.6.47 Util-linux-2.12q

El estándar FHS recomienda que se use /var/lib/hwclock para la ubicación del fichero adjtime, en lugar del habitual /etc. Para hacer que **hwclock** sea conforme a FHS, se ejecuta lo siguiente: [9]

```
sed -i 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
        hwclock/hwclock.c
mkdir -p /var/lib/hwclock
```

Util-linux falla al compilarse contra las nuevas versiones de Linux-Libc-Headers. El siguiente parche corrige de forma correcta dicho problema:

patch -Np1 -i ../util-linux-2.12q-cramfs-1.patch

Util-linux tiene un fallo de seguridad que podría permitir a un usuario remontar un volumen sin la opción nosuid. El siguiente parche corrige el problema:

patch -Np1 -i ../util-linux-2.12q-umount_fix-1.patch

Se prepara Util-linux para su compilación:

./configure

Se compila el paquete:

make HAVE_KILL=yes HAVE_SLN=yes

Significado de los parámetros de make:

HAVE_KILL=yes

Esto evita que el programa **kill** (que ya ha sido instalado por Procps) sea construido e instalado de nuevo.

HAVE_SLN=yes

Esto evita que el programa **sln** (un **ln** enlazado estáticamente, ya instalado por Glibc) se vuelva a construir e instalar.

Se instala el paquete y se mueve el binario **logger** a /bin, pues es necesario para los guiones de arranque:

make HAVE_KILL=yes HAVE_SLN=yes install

mv /usr/bin/logger /bin

6.6.48 Eliminar los símbolos de nuevo

Antes de hacer la eliminación de símbolos, se ha de tener mucho cuidado para asegurar que no se esté ejecutando ningún binario que vaya a ser procesado. [10]

Ahora se puede procesar con tranquilidad los binarios y librerías:

```
/tools/bin/find /{,usr/}{bin,lib,sbin} -type f \
        -exec /tools/bin/strip --strip-debug '{}' ';'
```

El directorio /tools ya puede ser borrado del sistema.

6.7 Sección 6

Esta sección detalla cómo instalar y configurar el paquete LFS-Bootscripts. Muchos de estos guiones funcionan sin necesidad de modificarlos, pero algunos necesitan ficheros de configuración adicionales, pues manejan información dependiente del hardware.

Se utilizan guiones de inicio al estilo System-V porque son ampliamente utilizados.

6.7.1 LFS-Bootscripts-3.2.1

El paquete LFS-Bootscripts contiene un conjunto de guiones para iniciar/parar el sistema durante el arranque/apagado. [10]

Se instala el paquete:

```
make install
```

6.7.2 Configuración del guión setclock

El guión **setclock** lee la hora del reloj interno del PC, conocido también como reloj BIOS o CMOS (Semiconductor de Oxido de Metal Complementario). Si el reloj hardware está establecido a la hora UTC, este guión la convierte a la hora local

mediante el fichero /etc/localtime (que le indica al programa **hwclock** en qué zona horaria se encuentra el usuario). No hay manera de detectar automáticamente si el reloj utiliza UTC o no, así que esto se debe configurar manualmente. [10]

Se crea un nuevo fichero /etc/sysconfig/clock ejecutando lo siguiente:

```
cat > /etc/sysconfig/clock << "EOF"
# Inicio de /etc/sysconfig/clock

UTC=0
# Fin de /etc/sysconfig/clock
EOF
```

6.7.3 Crear el fichero /etc/inputrc

El fichero /etc/inputrc se ocupa del mapeado del teclado para situaciones concretas. Este fichero es el fichero de inicio usado por Readline, la librería para cuestiones de entrada usada por Bash y otros intérpretes de comandos. [10]

Se puede utilizar un /etc/inputrc global genérico, con comentarios para explicar lo que hace cada opción. Se crea el fichero usando el siguiente comando:

```
cat > /etc/inputrc << "EOF"
# Inicio de /etc/inputrc

# Permite que la línea de comandos salte a la siguiente línea
set horizontal-scroll-mode Off

# Activa la entrada de 8 bits
set meta-flag On
set input-meta On
```

```
# Desactiva la supresión del bit 8
set convert-meta Off

# Mantiene el bit 8 para ser mostrado
set output-meta On

# none, visible o audible
set bell-style none

# Todo lo siguiente mapea la secuencia de escape
# del valor contenido en el primer argumento a las
# funciones específicas de readline

"\eOd": backward-word
"\eOc": forward-word

# Para la consola linux
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert

# Para xterm
"\eOH": beginning-of-line
"\eOF": end-of-line

# Para Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line
```

```
# Fin de /etc/inputrc
EOF
```

6.7.4 Configuración del guión localnet

Parte del trabajo del guión **localnet** es establecer el nombre de la máquina. Esto se configura en el fichero /etc/sysconfig/network. [10]

Se crea el fichero /etc/sysconfig/network y se introduce el nombre de la máquina ejecutando:

```
echo "HOSTNAME=localhost.localdomain" > /etc/sysconfig/network
```

6.7.5 Creación del fichero /etc/hosts

Se crea el fichero /etc/hosts ejecutando:

```
cat > /etc/hosts << "EOF"
# Inicio de /etc/hosts
127.0.0.1       localhost.localdomain       localhost

# Fin de /etc/hosts (versión sin tarjeta de red)
EOF
```

6.8. Sección 7

Es hora de hacer arrancable el sistema. En esta sección se explica la creación de un fichero fstab, la construcción de un núcleo para el nuevo sistema y la instalación del gestor de arranque GRUB para que el sistema se pueda seleccionar para arrancar al inicio.

6.8.1 Creación del fichero /etc/fstab

El fichero /etc/fstab lo utilizan ciertos programas para determinar dónde deben montarse los sistemas de ficheros, en qué orden y cuales deben comprobarse (por fallos de integridad) antes de montarse. Se crea una nueva tabla de sistemas de ficheros como esta: [10]

```
cat > /etc/fstab << "EOF"
# Inicio de /etc/fstab
# sistema de punto de tipo opciones volcado orden de
# ficheros   montaje                    chequeo
/dev/hda1    /       ext2 defaults    1      0
proc         /proc   proc defaults    0      0
sysfs        /sys    sysfs defaults   0      0
devpts       /dev/pts devpts gid=4,mode=620 0 0
shm          /dev/shm tmpfs defaults   0      0

# Fin de /etc/fstab
EOF
```

6.8.2 Linux-2.6.11.12

El paquete Linux contiene el núcleo Linux. Construir el núcleo comprende varios pasos: configuración, compilación e instalación. Se prepara la compilación ejecutando el siguiente comando:

```
make mrproper
```

Esto asegura que el árbol del núcleo está completamente limpio. El equipo del núcleo recomienda que se ejecute este comando antes de cada compilación del núcleo. No se puede confiar en que el árbol de las fuentes esté limpio tras desempaquetarlo.

Antes de iniciar la configuración es necesario aplicar un parche al núcleo, contenido en el paquete de RTAI, con la intención de agregarle soporte para acciones de tiempo real.

patch -p1 < /usr/src/rtai-3.3/base/arch/i386/patches/hal-linux-2.6.11-i386-r12.patch

Se configura el núcleo mediante una interfaz de menús.

make menuconfig

Algunas de las características del núcleo que se deben de terne en cuenta, para el correcto funcionamiento de RTAI, son las siguientes:

- **Loadable module support**: se activa “Enable module support”, “Module unloading”, y “Automatic module loading” y se desactiva “Module versioning support”.

- **Processor type and features:** se selecciona el tipo de subarquitectura como (PC-Compatible) y como familia de procesador un Pentium III, se activa ”Preemption Model”, se desactiva ”Use register arguments” y “Local APIC support on uniprocessors”.

- **Bus options:** se mantienen los valores por omisión.

- **Device Drivers:**

  - **Generic driver options:** se mantienen los valores por omisión.

  - **Memory Technology Devices (MTD):** no se necesita.

- **Parallel port support:** se desactiva. El puerto paralelo estándar no es util en experimentos de tiempo real; a través de los drivers de Comedi se puede acceder a este puerto.

- **Plug and Play support:** se mantienen los valores por omisión.

- **Block devices:** aquí se seleccionan los distintos dispositivos.

- **Network device support:** se mantienen los valores por omisión.

- **Input device support:** se asegura que Mouse esta seleccionado.

- **I2C support:** se desactiva; existen reportes de dificultades cuando se usan con RTAI.

- **Multimedia devices:** se desactiva.

- **Graphics support:** se mantiene desactivado.

- **Sound:** se desactiva

- **USB Support:** se desactiva.

- **File Systems:**

  - **Second extended fs support:** seleccionado

  - **Ext3 journalling file system support:** seleccionado y "Ext3 extended attributes"

  - **Reiserfs support:** no se necesita.

- **CD/ROM-DVD Filesystems:** no se selecciona.

- **DOS/FAT/NT Filesystems:** no se necesita.

Se compila la imagen del núcleo y los módulos:

make

Se instala los módulos:

make modules_install

Tras completar la compilación se necesitan algunos pasos adicionales para completar la instalación. Es necesario copiar varios ficheros al directorio /boot.

cp -v arch/i386/boot/bzImage /boot/lfskernel-2.6.11.12

System.map es un fichero de símbolos para el núcleo. Mapea los puntos de entrada de cada una de las funciones en la API del núcleo, al igual que las direcciones de las estructuras de datos del núcleo para el núcleo en ejecución. Se ejecuta el siguiente comando para instalar el fichero de mapa:

cp -v System.map /boot/System.map-2.6.11.12

.config es el fichero de configuración del núcleo creado por el paso **make menuconfig** anterior. Contiene todas las selecciones de configuración para el núcleo que se acaba de compilar. Es buena idea guardar este fichero como referencia futura:

cp -v .config /boot/config-2.6.11.12

6.8.3 RTAI

Antes de realizar la instalación de RTAI, es necesario crear un enlace simbólico a las fuentes del núcleo, con el siguiente comando: [11]

ln -s linux-2.6.11.12-adeos linux

La documentación de RTAI recomienda construirlo fuera del árbol de las fuentes:

mkdir rtai-3.3-builddir && cd rtai-3.3-builddir
make -f ../rtai-3.3/makefile srctree=../rtai-3.3 oldconfig

Se realiza la configuración:

make -f ../rtai-3.3/makefile srctree=../rtai-3.3 menuconfig

Se compila el paquete:

make

Se instala el paquete:

make install

Al instalar el paquete RTAI se crea el directorio /usr/realtime/ es importante agregar /usr/realtime/bin al PATH de variables en /etc/profile.

6.8.4 Comedi

Comedi provee los drivers, librerías de funciones y un API para interactuar con señales provenientes del hardware de adquisición de datos.

6.8.4.1 Comedilib

Este paquete contiene las librerías necesarias para el correcto funcionamiento de los driver de comedi.

Primero se configura el paquete:

./configure ––sysconfdir=/etc

Se compila el paquete:

make

Se instala el paquete:

make install

Se crea los dispositivos inodos en el directorio /dev/comedi[0-3].

make dev

6.8.4.2 Comedi

Se configura el paquete:

./configure --with-linuxdir=/usr/src/linux –with-rtaidir=/usr/realtime

Se compila el paquete:

make

Se instala el paquete:

make install

Se crean los dispositivos:

make dev

Algunos archivos hay que reubicarlos y hay que crear algunos enlaces para asegurar el correcto funcionamiento de los drivers.

cp include/linux/comedi.h /usr/include/
cp include/linux/comedilib.h /usr/include/

ln -s /usr/include/comedi.h /usr/include/linux/comedi.h
ln -s /usr/include/comedilib.h /usr/include/linux/comedilib.h
ln -s /usr/include/linux /usr/local/include/linux

6.8.5 Problemas con udev

Con la nueva plataforma de udev existen algunos problemas ya que los nuevos inodos (rtai, comedi) no están registrados; como se sabe udev crea automáticamente los dispositivos inodos en el directorio /dev una vez el dispositivo es detectado y luego son borrados cuando no estén en uso. [9]

RTAI usa inodos en /dev para los FIFO's, usados para el pase de mensajes entre el espacio usuario y el kernel, y para interactuar con el hardware de adquisición de datos, por lo tanto es necesario crear inodos persistentes, para lo cual se crea un ejecutable que cree estos inodos.

#rtai_inode: RTAI inode creation for UDEV systems, creates /dev/rtf(n)

```
for n in ‘ seq 0 9 ‘ ; do \
        mknod –m 666 / dev / rtf$n c 150 $n ; \
 done ; \
# create shared memory inode
mknod –m 666 / dev / rtai_shm c 10 254

# create Comedi inodes
for i in ‘ seq 0 15 ‘ ; do \
        rm / dev / comedi$i
        mknod –m 666 / dev / comedi$i c 98 $i \
         ; \
done ;
```

6.8.5 Hacer el sistema arrancable

El nuevo sistema está casi completo. Una de las últimas cosas por hacer es asegurarse de que puede ser arrancado.

Ahora se inicia el intérprete de comandos de **grub**:

```
grub
```

GRUB utiliza su propia estructura de nombres para los discos de la forma *(hdn,m)*, donde *n* es el número del disco duro y *m* es el número de la partición, comenzando ambos desde 0.
Se le indica a GRUB dónde debe buscar sus ficheros stage{1,2}.

```
root (hd0,1)
```

Se le indica a GRUB que se instale en el MBR de sda:

```
Setup (hd0,1)
```

GRUB informa que ha encontrado sus ficheros en /boot/grub. Esto es todo para activarlo. Se cierra el intérprete de comandos de **grub**:

quit

Se crea un fichero de "lista de menú" para definir el menú de arranque de GRUB:

```
cat > /boot/grub/menu.lst << "EOF"
# Inicio de /boot/grub/menu.lst
# Inicia por defecto la primera entrada del menú.
default 0
# Espera 30 segundos antes de iniciar la entrada por defecto.
timeout 30
# Colores
color green/black light-green/black
# Primera entrada
title Linux/RTAI
root (hd0,0)
kernel /boot/lfskernel-2.6.11.12 root=/dev/hda1
EOF
```

Capítulo 7

Desarrollo de Aplicaciones

En este capítulo de desarrollan algunas aplicaciones que verifican el correcto funcionamiento de RTAI, se comprueban alguna de las características de RTAI y finalmente se realiza una aplicación de control de procesos para experimentar con el PC104 como dispositivo de control en tiempo real.

7.1 Introducción

Sobre la plataforma de tiempo real construida se pueden realizar un gran numero de aplicaciones, que permitan comprobar le rendimiento del sistema Linux/RTAI sobre el PC104. La necesidad de incorporar a este estándar de PC en los procesos industriales actuales, le exigen una función con vigentes y versátiles herramientas de automatización que le permitan estar al nivel de algunos autómatas programables. Las aplicaciones desarrolladas permiten verificar el funcionamiento de algunos de los mecanismos de comunicación de RTAI como lo son los FIFOs y Semáforos, también se lograron probar algunas aplicaciones que presentan al PC104 como herramienta de control, sobre algunos procesos simulados en un PC personal.

7.2 Prueba de los FIFOS

Los FIFOS permiten el paso de mensajes entre tareas y programas en tiempo real que pueden estar en el espacio usuario.

La aplicación consiste en colocar una señal dentro de un FIFO y poder leerla desde otra tarea.

Primero se crea un generador de señales que envía información a un FIFO, como lo muestra en la siguiente figura: [8]

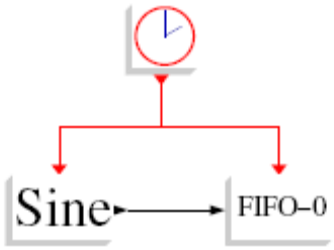

Figura 28: Señal senoidal dentro de un FIFO

Luego se crea un programa capaz de leer la información contenida en el FIFO seleccionado por el diagrama anterior.

```
/ _ readfifo.c –– Read a FIFO and printits data_/
#include <stdio.h>
#include <unitd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
Static int end ;
struct data {
        float t ;
        float u [ 1 ] ;
};
```

```
static void do_end ( int dummy ) { end = 1; }
int main ( void )
{
        int fifo ;
        struct data val ;

        i f ( ( fifo = open ( " / dev/rtf0 " , O_RDONLY ) ) < 0) {
                fprintf ( stderr , " Error opening /dev/rtf0 \n " ) ;
                exit ( 1 ) ;
        }

        signal ( SIGINT , do_end ) ;

        while ( ! end ) {
                read ( fifo , &val , sizeof ( val ) ) ;
                printf ( "%f \ t%f \n " , val . t , val . u [ 0 ] ) ;}

        return 0;
}
```

Ahora se puede correr el programa contenido en el diagrama desde el PC104, luego corremos el programa readfifo.c en la PC personal, la cual se encuentra conectada vía red con el PC104 y finalmente utilizamos el osciloscopio que proporciona **rtailab** compatible con el sistema Scilab para poder visualizar el contenido del FIFO. [8]

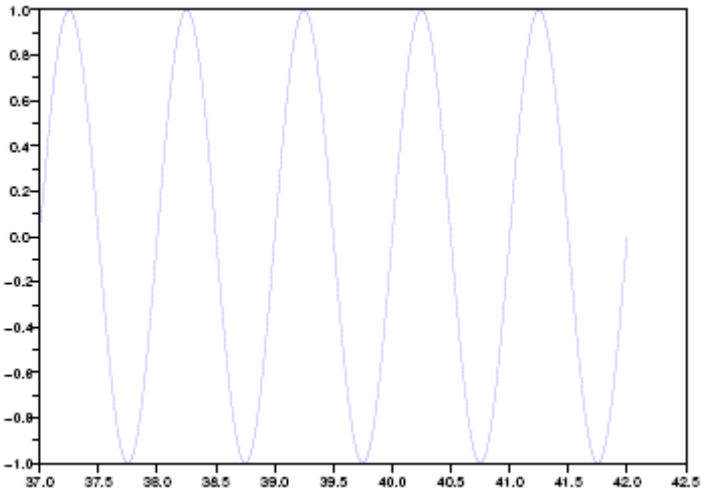

Figura 29: Contenido del FIFO

Esta aplicación permite verifcar el correcto funcionamiento de los FIFOS, ya que estos son muy importantes para el pase de mensajes en tiempo real entre tareas que pueden estar ejecutándose en el espacio usuario.

7.3 Prueba de los Semáforos

Los semáforos son usados como métodos de señalización o sincronización entre procesos de tiempo real. Para verificar su funcionamiento se implementa el diagrama de la siguiente figura. [8]

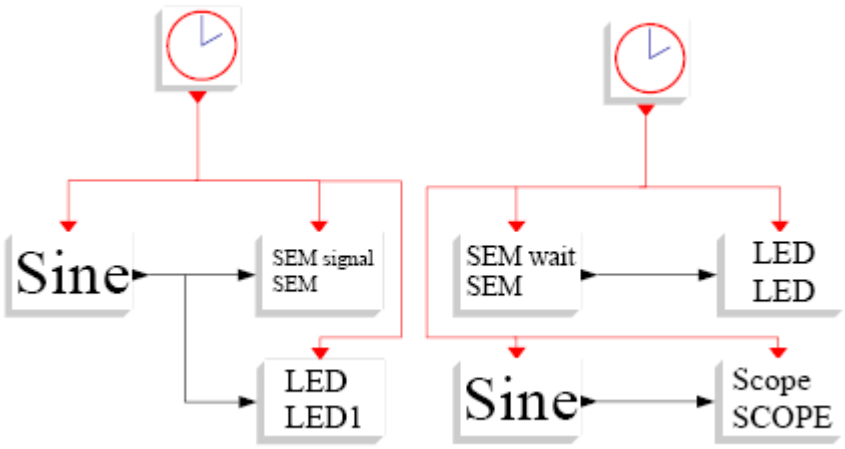

Figura 30: Uso de los Semáforos

Como se muestra en el diagrama, la aplicación consiste en 2 tareas que usan los semáforos como medio de sincronización, la tarea de la izquierda utiliza una señal senoidal para modificar el estado del semáforo, la tarea de la derecha espera mientras lee el semáforo y produce una salida solo cuando la onda senoidal en la tarea de señalización es positiva.

Al correr esta aplicación dentro del PC104, se puede observar desde el osciloscopio de la PC personal la siguiente salida, que verifica el correcto funcionamiento de los semáforos.

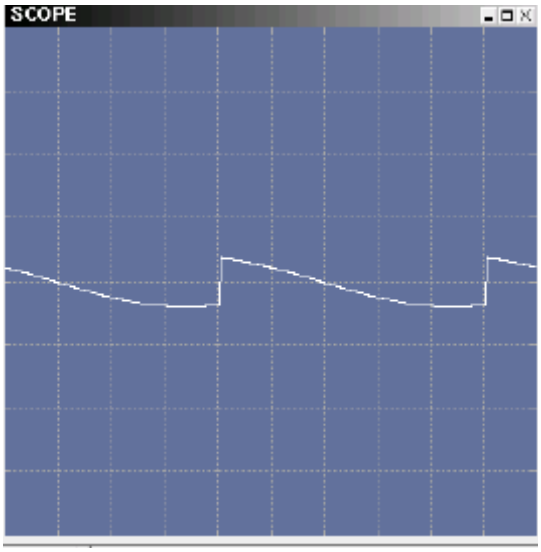

Figura 31: Salida del Osciloscopio

7.4 Control de un Motor DC

En esta aplicación se muestra como se puede adaptar la programación grafica con RTAI y el uso del PC104 como herramienta de control. Este ejemplo esta basado en el tutorial de control de Matlab y Simulink "Modelado de la posición del Motor DC en Simulink" sección 7.2. [8]

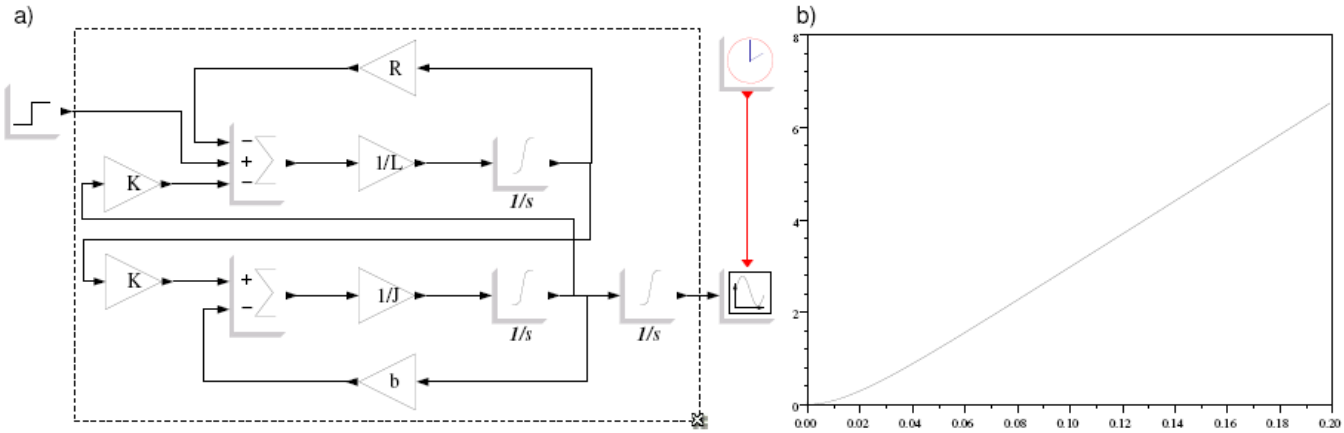

Figura 32: La parte a) muestra el modelo de la posición del motor y la parte b) muestra la salida (posición) del motor ante un escalón unitario

La función de transferencia discreta del motor esta dada por la siguiente ecuación:

$$\frac{0.001z + 0.001}{z^2 - 1.9425z + 0.9425}$$

Tomando en cuenta las variables J = 3.2284E-6; b = 3.5077E-6; K = 0.0274, R = 4; L = 2.75E-6.

Aplicando un controlador discreto y cerrando el lazo del sistema, se obtiene el siguiente diagrama:

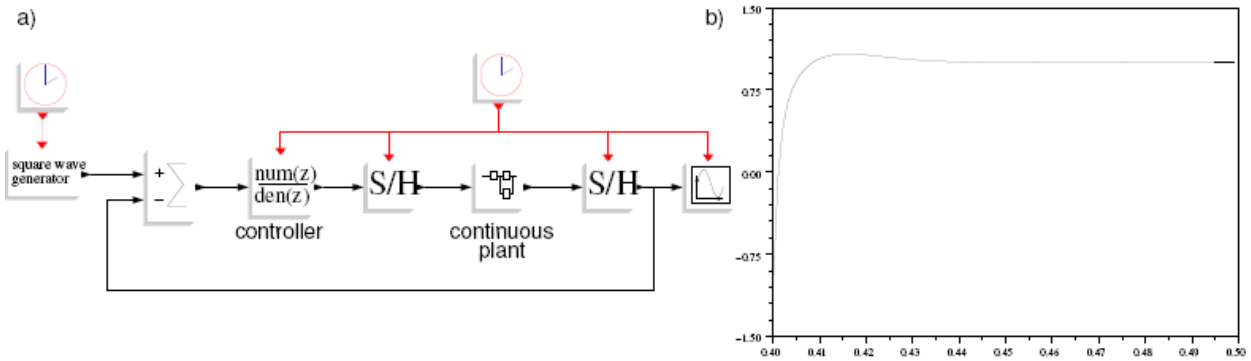

Figura 33: La parte a) muestra el diagrama del sistema controlado y la parte b) muestra la salida del sistema (posición) ante un escalón unitario.

Ahora se implementa el controlador bajo RTAI, lo cual origina el siguiente diagrama de bloques:

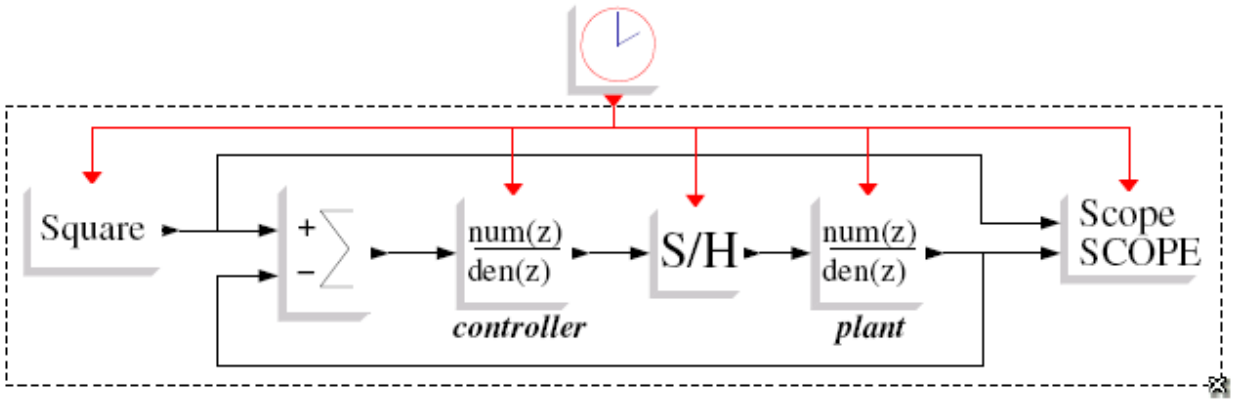

Figura 34: Diagrama de bloques del sistema controlado bajo RTAI.

La función de transferencia del controlador es la siguiente:

Numerador: $450*z^2 - 765*z + 325.125$
Denominador: $z^2 + 0.28*z - 0.686$

Finalmente se corre esta aplicación sobre el PC104 y por medio de la conexión establecida con el PC personal, se puede observar en el osciloscopio la señal del sistema controlado.

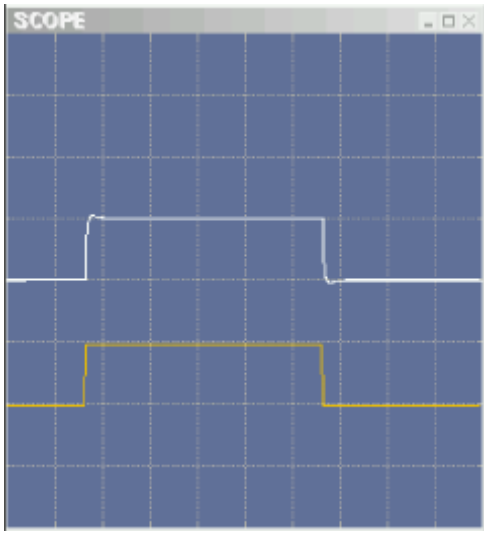

Figura 35: Salida del Sistema Controlado

7.5 Sistema de Control

En esta aplicación se trabaja con un sistema de control en tiempo real, el cual consiste en conectar el PC104 a un PC personal, donde se simula el comportamiento de una planta real, por medio del puerto serial. En la siguiente figura se muestra el diagrama de bloques del sistema.

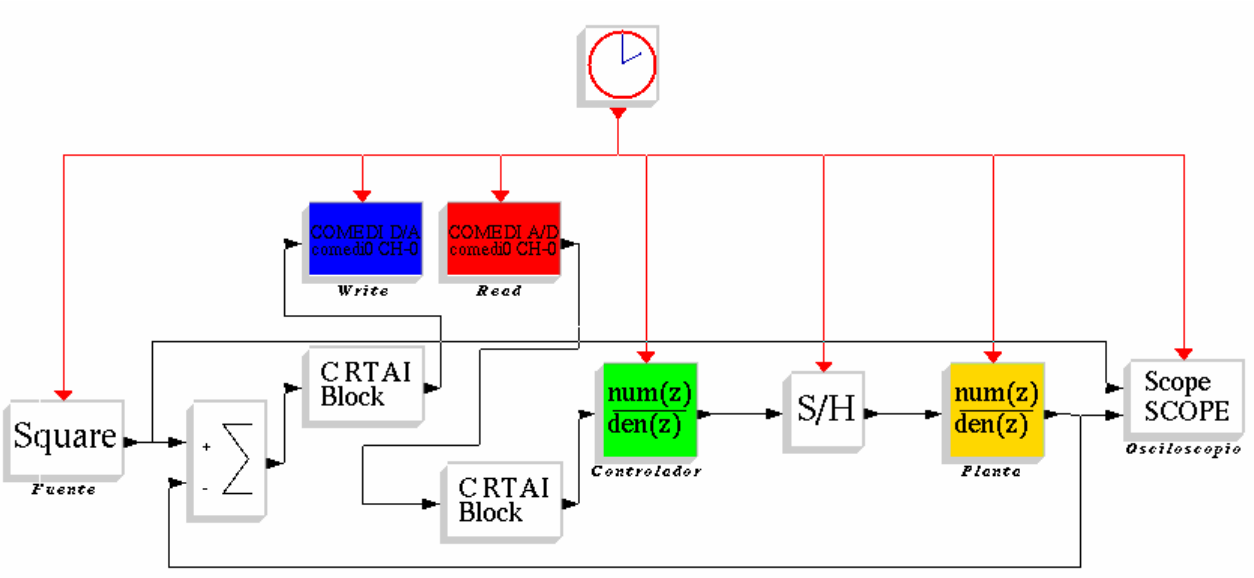

Figura 36. Sistema de Control.

Como se observa en la figura, se tiene el modelo de una planta real en este caso el mismo modelo del motor DC de la aplicación anterior, el cual es controlado desde el PC104, es decir, los coeficientes del controlador son asignados por una rutina en el PC104 a través del puerto serial.

El bloque amarillo representa la planta (motor), el bloque verde representa al controlador, el bloque azul representa la escritura en el puerto serial o envío de información, el bloque rojo representa la lectura del puerto serial.

El código del diagrama de bloques se puede observar en el Anexo C a esta rutina solo hay que agregarle unas cuantas líneas de código para poder acceder a la información en el puerto serial; dentro del bloque de escritura del puerto podemos encontrar unas líneas similares a las siguientes:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>
#include <rtai_serial.h>
#include <rtai_lxrt.h>
#define PERIOD 15625000LL
#define MS100 100000000LL
#define BAUD 9600
#define NUMBITS 8
#define STOPBITS 1
#define BUFLEN 4

int senddata(int fd) {
        unsigned char buf[BUFLEN];
        buf[0] = 0x43;
        buf[1] = 0x41;
        buf[2] = 0x50;
        buf[3] = 0x00;
        if (rt_spwrite(fd, buf, sizeof(buf))) { return 0;   }
        return 1;
}
```

Dentro del bloque de lectura del puerto podemos encontrar unas líneas similares a las siguientes:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#include <rtai_serial.h>
#include <rtai_lxrt.h>
#define PERIOD 15625000LL
#define MS100 100000000LL
#define BAUD 9600
#define NUMBITS 8
#define STOPBITS 1
#define BUFLEN 4

int receivedata(int fd) {
        unsigned char buf[BUFLEN] = {0};
        if(rt_spread_timed(fd, &buf[0], BUFLEN, nano2count(MS100))) {
        return 0;
}
        printf("serial data=%s\n", buf);
        return 1;
}
```

La rutina que se emplea en el PC104 para la asignación de los coeficientes del controlador, corresponde al siguiente diagrama:

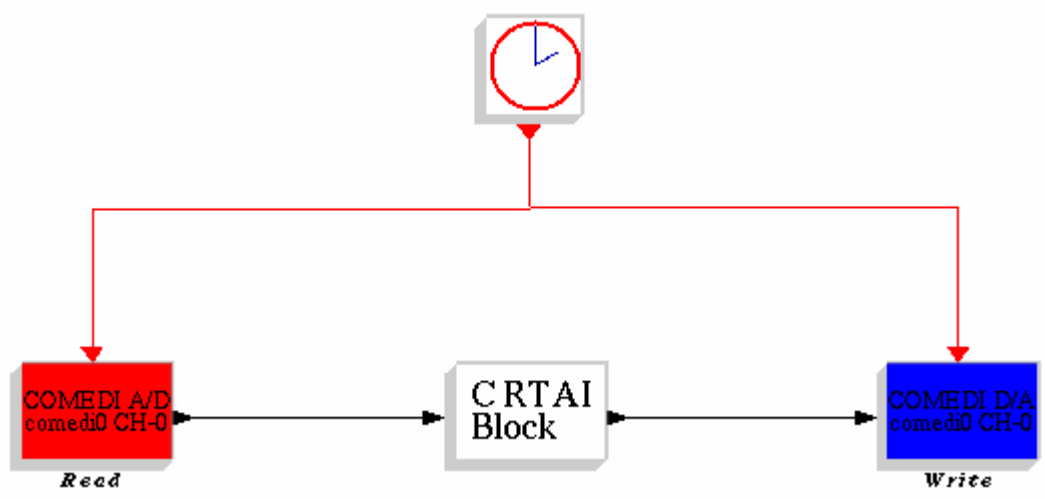

Figura 37. Diagrama de Bloques de la Rutina Para El PC104.

En realidad no se ha utilizado ningún tipo de algoritmo para el diseño del controlador, solo se han asignado los valores correspondientes al diseño realizado en el manual de Matlab para el control del motor DC. La principal utilidad de esta aplicación es demostrar la comunicación entre el PC104 y un PC personal en tiempo real, abriendo la posibilidad de aplicar distintas técnicas de control sobre cualquier modelo simulado o sistema físico.

La respuesta del sistema ante una entrada escalón, se puede observar en el osciloscopio virtual que suministra la herramienta **rtailab**, y por supuesto coincide con la salida de la aplicación anterior.

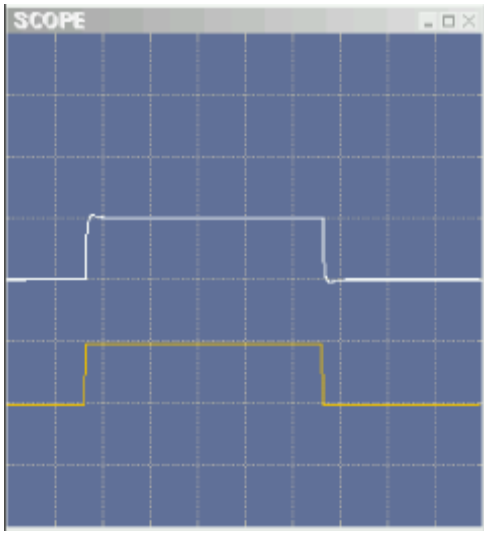

Figura 35.

Capítulo 8

Conclusiones

Los sistemas empotrados corresponden a uno de los dispositivos mas utilizados para la supervisón y monitorización de distintos procesos físicos en donde la adversidad, la posible repetición de rutinas y las exigentes condiciones físicas limitan el rango de posibilidades; esta tendencia a implementar estos sistemas en los actuales procesos físicos de la industria, aumenta la demanda de sistemas operativos capaces de ofrecer un rendimiento optimo y mínimo uso de los recursos disponibles, esto genera una gran oferta de sistemas de tiempo real, en su mayoría con licencias algo costosas, lo cual le ha abierto el campo a los sistemas basados en software libre y ha despertado el interés de los desarrolladores de software y hardware por este tipo de sistemas, cuyo trabajo en conjunto a logrado completar herramientas como RTAI, RTLinux, Kurt, etc.

Entre uno de los sistemas empotrados mas destacados se encuentra el PC104, cuya arquitectura común, le permite una compatibilidad con muchas de las aplicaciones y dispositivos diseñados para equipos de escritorio, por lo que este estudio se baso en lograr construir un sistema de tiempo real basado en software libre capaz de explotar las ventajas que posee este dispositivo. La construcción de un sistema operativo desde cero permite una entera manipulación de las distintas herramientas que se requieren dentro del sistema, lo cual puede considerarse como un uso adecuado de los recursos ya que en estos sistemas empotrados las limitaciones de memoria es una de las principales características a tener en cuenta, por lo que el diseño y construcción del sistema debe ajustarse a estos limites.

Al tener en funcionamiento el PC104 con un sistema de tiempo real, es posible utilizar este dispositivo como herramienta de control sobre distintos sistemas físicos, que requieran una posible supervisión o monitorización remota. Gracias al estándar sobre el cual esta construido el PC104 y al sistema operativo que lo rige, este puede alcanzar un alto rango de compatibilidad con varios de los protocolos de comunicación como TCP/IP, DNP3, RS-232, por medio de puertos ethernet, serial, paralelo, usb, etc.

La compatibilidad de RTAI con el código C, permite la creación de sencillas aplicaciones en tiempo real, que pueden encargarse de manejar distintas interrupciones del sistema. Rtailab representa una poderosa herramienta para la implementación de aplicaciones de control en el ambiente de Linux/Rtai.

Bibliografía

[1] MOPSlcd7 Manual de Usuario. http://www.kontron.com.

[2] PCM-3116 Manual de Usuario. http://www.tri-m.com.

[3] PCM-3660 Manual de Usuario. http://www.tri-m.com.

[4] Xtreme/104 Manual de Usuario. http://www.connect.tech.com.

[5] PC/104 Vehicle Power Supply. Manual de Usuario. http://www.tri-m.com.

[6] VT104. http://www.tri-m.com.

[7] RTAI 3.3 Manual del Usuario, Paolo Mantegazza, 2006. http://www.rtai.org.

[8] RTAI-Lab tutorial: Scilab, Comedi, and real-time control, Roberto Bucher, 2006.

[9] Más Allá de Linux From Scratch: Versión svn-20050428, 2004.

[10] Linux From Scratch, Version 6.1.1, Gerard Beekmans, 2006.

[11] RTAI Guia de Principiante, Paolo Mantegazza, http://www.rtai.org.

[12] Análisis de Sistemas Operativos de Tiempo Real Libres, Zamarrón López, 2004. http://www.ciclope.info/doc/rtos/select.php

[13] Introducción a Scilab, Prof. Paulo Sergio da Motta Pires, Universidade Federal de Rió Grande del Norte, 2005, Brasil.

[14] Linux Empotrado, Gaspoz Frédéric, HEVS Sion section électricité-infotronique, 2002.

[15] PC104 Especificaciones, PC/104 Embedded Consortium, Versión 2.5, Noviembre 2003.

[16] Porting RTAI over Adeos, Philippe Gerum, http://www.aero.polimi.it/~rtai/.

[17] (RTAI) in low cost high performance motion control, Paolo Mantegazza, Dipartimento di Ingegneria Aerospaziale Politecnico di Milano, 2003.

[18] Introduction to Linux for Real-Time Control, Sarah Bishop, National Institute of Standards and Technology, 2004.

[19] Sistemas Inteligentes en Tiempo Real, Vicente J. Botti Navarro, Universidad Politécnica de Valencia, 2002.

[20] CACSD with Linux RTAI and RTAI-Lab, R. Bucher and L. Dozio, in Real Time Linux Workshop, Valencia, 2003.

[21] http://es.wikipedia.org/wiki/Sistema_integrado.

Anexo A

Comprobando El Desempeño De RTAI

Una vez completada la instalación de RTAI es recomendable probar los test de rendimiento que el paquete instala en el directorio /usr/realtime/testsuite/kern. El primer script que se ejecuta se encuentra en /usr/realtime/testsuite/kern/latency el cual mide la latencia y jitter de la maquina, todos los resultados mostrados por el test de latencia son en nanosegundos los cuales se muestran a continuación: [7]

```
rtai:/usr/realtime/testsuite/kern/latency # ./run
*
*
* Type ^C to stop this application.
*
*

## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot

RTAI Testsuite - KERNEL latency (all data in nanoseconds)
RTH|    lat min|    ovl min|    lat avg|    lat max|    ovl max|   overruns
RTD|      -2057|      -2057|       3184|       9850|       9850|          0
RTD|      -2020|      -2057|       3180|       8544|       9850|          0
RTD|      -2013|      -2057|       3171|       8427|       9850|          0
RTD|      -2029|      -2057|       3158|       8621|       9850|          0
RTD|      -2037|      -2057|       3177|       7842|       9850|          0
RTD|      -2035|      -2057|       3159|       8346|       9850|          0
RTD|      -2020|      -2057|       3181|       7530|       9850|          0
RTD|      -2055|      -2057|       3183|       8668|       9850|          0
RTD|      -2030|      -2057|       3152|       8226|       9850|          0
RTD|      -2026|      -2057|       3156|       8080|       9850|          0
RTD|      -2051|      -2057|       3214|       8307|       9850|          0
RTD|      -2019|      -2057|       3204|       7639|       9850|          0
RTD|      -2031|      -2057|       3274|       7733|       9850|          0
```

El siguiente script corresponde a los **switches**, el cual puede proporcionar cierta información respecto al máximo tiempo que necesita RTAI para atender las interrupciones; este script se encuentra en /usr/realtime/testsuite/kern/switches cuyo resultado es el siguiente: [7]

```
(none):/usr/realtime/testsuite/user/switches # ./run
*
*
* Type ^C to stop this application.
*
*

Wait for it ...

FOR 10 TASKS: TIME 15 (ms), SUSP/RES SWITCHES 20000, SWITCH TIME 789 (ns)

FOR 10 TASKS: TIME 17 (ms), SEM SIG/WAIT SWITCHES 20000, SWITCH TIME 851 (ns)

(none):/usr/realtime/testsuite/user/switches #
```

El último test corresponde al **preempt** el cual esta diseñado para probar los planificadores bajo una intensa carga; esta utilidad se puede ver como una utilidad de stress. Este software combina la tarea de calibración de latencia con una tarea rápida y otra lenta que tiene 2 niveles de prioridad anidando un número impar de tareas. [7]

Para verificar mejor el funcionamiento de este script se puede aplicar stress adicional a al maquina y luego iniciar el script para verificar los tiempos de latencia y jitter.

Anexo B

Disposición de Conexiones del MOPSlcd7

La configuración de conexiones disponibles para este modulo son las siguientes: [1]

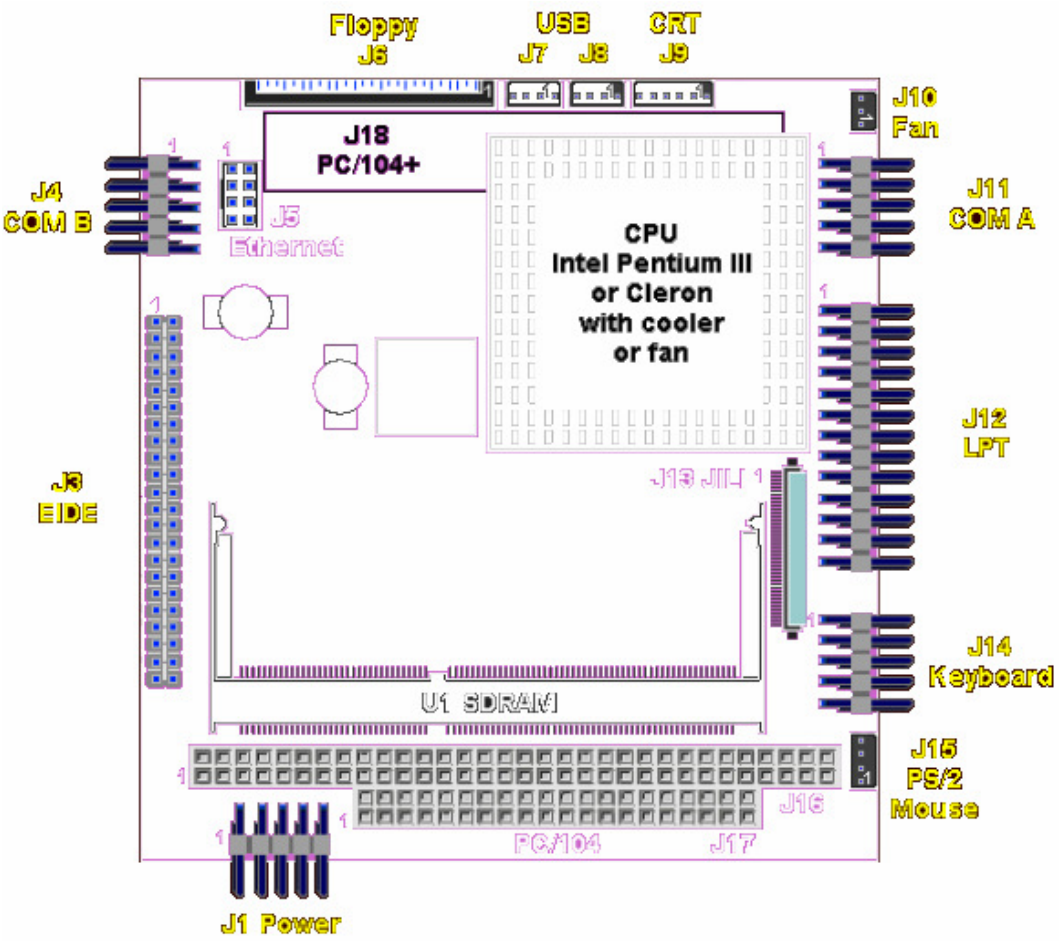

Figura 38. Disposición de Conexiones del MOPSlcd7

Cada conector significa:

Connector	Function
J1	Power Connector
J3	IDE Hard Disk Connector
J4, J11	Serial Interfaces COM A and COM B Connectors
J5	Ethernet Interface Connector
J6	Floppy Drive Connector
J7, J8	USB interface Connectors
J9	CRT Monitor Connector
J10	Fan Connector
J12	LPT Connector
J13	Flat-panel display Connector
J14	Keyboard and Feature Connector
J15	PS/2 Mouse Connector
J16	PC/104 Bus (XT-bus part)
J17	PC/104 Bus (AT-bus part)
J18	PC/104-Plus Bus (PCI part)
U1	SDRAM

Anexo C

Código de la Aplicación 7.3 Motor DC

```
#include "/usr/local/scilab-4.0/routines/machine.h"
#include "/usr/local/scilab-4.0/routines/scicos/scicos_block.h"
#include <math.h>
#include <memory.h>

/* Code prototype for standalone use  */
/*     Generated by Code_Generation toolbox of Scicos with scilab-4.0 */
/*     date : 16-Oct-2006 */

void dcmotmain0(scicos_block *block_dcmot,double *z, double *t,
  int phase);

void dcmotmain1(scicos_block *block_dcmot,double *z, double *t);

void dcmotmain2(scicos_block *block_dcmot,double *z, double *t) ;

void dcmot_init(scicos_block *block_dcmot,double *z, double *t) ;

void dcmot_end(scicos_block *block_dcmot,double *z, double *t) ;

int dcmot_init_blk() ;

int dcmot_rt_exec(scicos_block *block_dcmot,double *z, double *t);

int dcmotddoit1(double *, double *, double *, int *);

int dcmotodoit(double *, double *, int* , int );

int dcmotddoit2(double *, double *, double *, int *);

int dcmotedoit1(double *, double *, int *, int *);

int dcmotozdoit(double *,double *, int*, int *, int);

int dcmot_initi(double *, double *, int *);

int dcmot_endi(double *, double *, int *);
```

```
int dcmot_outtb(double *, double *, int *);

int dcmot_putevs(double *, int *, int *);

void dcmot_events(int *nevprt, double *t);

static void set_nevprt(int nevprt);

void o_scope_SCOPE(scicos_block *, int );

void C2F(samphold)(int *, int *, double *, double *, double *, int *,
  double *, int *, double *, int *, double *, int *,int *,int *,
  double *, int *, double *, int *);

void C2F(dsslti)(int *, int *, double *, double *, double *, int *,
  double *, int *, double *, int *, double *, int *,int *,int *,
  double *, int *, double *, int *);

void summation(scicos_block *, int );

void i_square_14(scicos_block *, int );
/* Table of constant values */
static  integer nrd_0 = 0;
static  integer nrd_1 = 1;
static  integer nrd_2 = 2;
static  integer nrd_3 = 3;
static  integer nrd_4 = 4;
static  integer nrd_5 = 5;
static  integer nrd_6 = 6;
static  integer nrd_7 = 7;
static  integer nrd_8 = 8;
static  integer nrd_9 = 9;

static integer totalnevprt;
static integer evtspt[ ]={-1};
static double zero=0;
static integer clkptr[ ]={1,1,1,1,1,1,1,2};
static integer ordptr[ ]={1,7};
static double tevts[ ] = {0};
static integer pointi={0};
static double w[1];
double *dcmot_block_outtb;
static int *dcmot_iwa;
static int aaa=0, bbb=0;
static void set_nevprt(int nevprt)
{
  totalnevprt=nevprt;
}

scicos_block block_dcmot[7];

static double RPAR1[ ] = {
/* Routine name of block: dsslti
   Gui name of block: DLR_f
Label: Controlador
Exprs: 450*z^2-765*z+325.125
```

```
Identification: C o n t r o l a d o r
rpar= {0,-0.9425,1,1.9425,0,1,-99,109.125,450};
*/
0,-0.9425,1,1.9425,0,1,-99,109.125,450,
/* Routine name of block: dsslti
   Gui name of block: DLR_f
Exprs: 0.001*z+0.001
Identification: P l a n t a
rpar= {0,-0.94525,1,1.94525,0,1,0.001,0.001,0};
*/
0,-0.94525,1,1.94525,0,1,0.001,0.001,0,
/* Routine name of block: i_square_14
   Gui name of block: rtai_square
Exprs: 1,4,2,0,0
Identification: F u e n t e
rpar= {1,4,2,0,0};
*/
1,4,2,0,0,
};

static integer NRPAR1 = 3;
static integer NTOTRPAR1 = 23;
char * strRPAR1[3] = {"C o n t r o l a d o r","P l a n t a","F u e n t e"};
int lenRPAR1[3] = {9,9,5};

static integer IPAR1[ ] = {
/* Routine name of block: summation
 Gui name of block: SUMMATION
Compiled structure index: 5
Exprs: [1;-1]
Identification: IPARAM[1]
ipar= {1,-1};
*/
1,-1,
/* Routine name of block: bidon
 Gui name of block: EVTGEN_f
Compiled structure index: 7
Exprs: 1
Identification: IPARAM[2]
ipar= {1};
*/
1,
};
static integer NIPAR1 = 2;
static integer NTOTIPAR1 = 3;
char * strIPAR1[2] = {"IPARAM[1]","IPARAM[2]"};
int lenIPAR1[2] = {2,1};

double z[]={0,0,0,0,0,0,0,0,0,0,0,0,0};
/* Note that z[]=[z_initial_condition;outtb;iwa]
z_initial_condition= {0,0,0,0,0,0};
outtb= {0,0,0,0,0};
iwa= {0,0};
work= {0,0,0,0,0,0,0};
*/
```

```
double get_tsamp()
{
  return(0.01);
}

#include "dcmot_io.c"

/*---------------------------------------- Scicos blocks initialisation */
int
dcmot_init_blk()
{
  double t;
  int nevprt=1;

  void **work;
  work = (void **) (z+13);
  dcmot_block_outtb = z+6;
  dcmot_iwa = (int *)(z+11);

/* Routine name of block: o_scope_SCOPE
 Gui name of block: rtai_scope
Compiled structure index: 1
Exprs: 2,SCOPE
Identification: O s c i l o s c o p i o
z= {0};
*/

/* Routine name of block: dsslti
 Gui name of block: DLR_f
Compiled structure index: 3
Label: Controlador
Exprs: 450*z^2-765*z+325.125
Identification: C o n t r o l a d o r
z= {0,0};
*/

/* Routine name of block: dsslti
 Gui name of block: DLR_f
Compiled structure index: 4
Exprs: 0.001*z+0.001
Identification: P l a n t a
z= {0,0};
*/

/* Routine name of block: i_square_14
 Gui name of block: rtai_square
Compiled structure index: 6
Exprs: 1,4,2,0,0
Identification: F u e n t e
z= {0};
*/
  block_dcmot[0].type = 2004;
  block_dcmot[0].ztyp = 0;
  block_dcmot[0].ng = 0;
  block_dcmot[0].nz = 1;
  block_dcmot[0].nrpar = 0;
```

```
block_dcmot[0].nipar = 0;
block_dcmot[0].nin = 2;
block_dcmot[0].nout = 0;
block_dcmot[0].nevout = 0;
block_dcmot[0].nmode = 0;
if ((block_dcmot[0].insz=malloc(sizeof(int)*block_dcmot[0].nin))== NULL ) return 0;
if ((block_dcmot[0].inptr=malloc(sizeof(double*)*block_dcmot[0].nin))== NULL ) return 0;
if ((block_dcmot[0].evout=calloc(block_dcmot[0].nevout,sizeof(double)))== NULL )return 0;
block_dcmot[0].inptr[0] = &(dcmot_block_outtb[1]);
block_dcmot[0].insz[0] = 1;
block_dcmot[0].inptr[1] = &(dcmot_block_outtb[0]);
block_dcmot[0].insz[1] = 1;
if ((block_dcmot[0].outsz=malloc(sizeof(int)*block_dcmot[0].nout))== NULL ) return 0;
if ((block_dcmot[0].outptr=malloc(sizeof(double*)*block_dcmot[0].nout))== NULL ) return 0;
block_dcmot[0].z=&(z[0]);
block_dcmot[0].work=(void **)(((double *)work)+0);
block_dcmot[1].type = 0;
block_dcmot[1].ztyp = 0;
block_dcmot[1].ng = 0;
block_dcmot[1].nz = 0;
block_dcmot[1].nrpar = 0;
block_dcmot[1].nipar = 0;
block_dcmot[1].nin = 1;
block_dcmot[1].nout = 1;
block_dcmot[1].nevout = 0;
block_dcmot[1].nmode = 0;
if ((block_dcmot[1].insz=malloc(sizeof(int)*block_dcmot[1].nin))== NULL ) return 0;
if ((block_dcmot[1].inptr=malloc(sizeof(double*)*block_dcmot[1].nin))== NULL ) return 0;
if ((block_dcmot[1].evout=calloc(block_dcmot[1].nevout,sizeof(double)))== NULL )return 0;
block_dcmot[1].inptr[0] = &(dcmot_block_outtb[2]);
block_dcmot[1].insz[0] = 1;
if ((block_dcmot[1].outsz=malloc(sizeof(int)*block_dcmot[1].nout))== NULL ) return 0;
if ((block_dcmot[1].outptr=malloc(sizeof(double*)*block_dcmot[1].nout))== NULL ) return 0;
block_dcmot[1].outptr[0]=&(dcmot_block_outtb[3]);
block_dcmot[1].outsz[0]=1;
block_dcmot[1].z=&(z[1]);
block_dcmot[1].work=(void **)(((double *)work)+1);
block_dcmot[2].type = 0;
block_dcmot[2].ztyp = 0;
block_dcmot[2].ng = 0;
block_dcmot[2].nz = 2;
block_dcmot[2].nrpar = 9;
block_dcmot[2].nipar = 0;
block_dcmot[2].nin = 1;
block_dcmot[2].nout = 1;
block_dcmot[2].nevout = 0;
block_dcmot[2].nmode = 0;
if ((block_dcmot[2].insz=malloc(sizeof(int)*block_dcmot[2].nin))== NULL ) return 0;
if ((block_dcmot[2].inptr=malloc(sizeof(double*)*block_dcmot[2].nin))== NULL ) return 0;
if ((block_dcmot[2].evout=calloc(block_dcmot[2].nevout,sizeof(double)))== NULL )return 0;
block_dcmot[2].inptr[0] = &(dcmot_block_outtb[4]);
block_dcmot[2].insz[0] = 1;
if ((block_dcmot[2].outsz=malloc(sizeof(int)*block_dcmot[2].nout))== NULL ) return 0;
if ((block_dcmot[2].outptr=malloc(sizeof(double*)*block_dcmot[2].nout))== NULL ) return 0;
block_dcmot[2].outptr[0]=&(dcmot_block_outtb[2]);
block_dcmot[2].outsz[0]=1;
```

```
block_dcmot[2].z=&(z[1]);
block_dcmot[2].rpar=&(RPAR1[0]);
block_dcmot[2].work=(void **)(((double *)work)+2);
block_dcmot[3].type = 0;
block_dcmot[3].ztyp = 0;
block_dcmot[3].ng = 0;
block_dcmot[3].nz = 2;
block_dcmot[3].nrpar = 9;
block_dcmot[3].nipar = 0;
block_dcmot[3].nin = 1;
block_dcmot[3].nout = 1;
block_dcmot[3].nevout = 0;
block_dcmot[3].nmode = 0;
if ((block_dcmot[3].insz=malloc(sizeof(int)*block_dcmot[3].nin))== NULL ) return 0;
if ((block_dcmot[3].inptr=malloc(sizeof(double*)*block_dcmot[3].nin))== NULL ) return 0;
if ((block_dcmot[3].evout=calloc(block_dcmot[3].nevout,sizeof(double)))== NULL )return 0;
block_dcmot[3].inptr[0] = &(dcmot_block_outtb[3]);
block_dcmot[3].insz[0] = 1;
if ((block_dcmot[3].outsz=malloc(sizeof(int)*block_dcmot[3].nout))== NULL ) return 0;
if ((block_dcmot[3].outptr=malloc(sizeof(double*)*block_dcmot[3].nout))== NULL ) return 0;
block_dcmot[3].outptr[0]=&(dcmot_block_outtb[0]);
block_dcmot[3].outsz[0]=1;
block_dcmot[3].z=&(z[3]);
block_dcmot[3].rpar=&(RPAR1[9]);
block_dcmot[3].work=(void **)(((double *)work)+3);
block_dcmot[4].type = 4;
block_dcmot[4].ztyp = 0;
block_dcmot[4].ng = 0;
block_dcmot[4].nz = 0;
block_dcmot[4].nrpar = 0;
block_dcmot[4].nipar = 2;
block_dcmot[4].nin = 2;
block_dcmot[4].nout = 1;
block_dcmot[4].nevout = 0;
block_dcmot[4].nmode = 0;
if ((block_dcmot[4].insz=malloc(sizeof(int)*block_dcmot[4].nin))== NULL ) return 0;
if ((block_dcmot[4].inptr=malloc(sizeof(double*)*block_dcmot[4].nin))== NULL ) return 0;
if ((block_dcmot[4].evout=calloc(block_dcmot[4].nevout,sizeof(double)))== NULL )return 0;
block_dcmot[4].inptr[0] = &(dcmot_block_outtb[1]);
block_dcmot[4].insz[0] = 1;
block_dcmot[4].inptr[1] = &(dcmot_block_outtb[0]);
block_dcmot[4].insz[1] = 1;
if ((block_dcmot[4].outsz=malloc(sizeof(int)*block_dcmot[4].nout))== NULL ) return 0;
if ((block_dcmot[4].outptr=malloc(sizeof(double*)*block_dcmot[4].nout))== NULL ) return 0;
block_dcmot[4].outptr[0]=&(dcmot_block_outtb[4]);
block_dcmot[4].outsz[0]=1;
block_dcmot[4].z=&(z[5]);
block_dcmot[4].ipar=&(IPAR1[0]);
block_dcmot[4].work=(void **)(((double *)work)+4);
block_dcmot[5].type = 2004;
block_dcmot[5].ztyp = 0;
block_dcmot[5].ng = 0;
block_dcmot[5].nz = 1;
block_dcmot[5].nrpar = 5;
block_dcmot[5].nipar = 0;
block_dcmot[5].nin = 0;
```

```
  block_dcmot[5].nout = 1;
  block_dcmot[5].nevout = 0;
  block_dcmot[5].nmode = 0;
  if ((block_dcmot[5].insz=malloc(sizeof(int)*block_dcmot[5].nin))== NULL ) return 0;
  if ((block_dcmot[5].inptr=malloc(sizeof(double*)*block_dcmot[5].nin))== NULL ) return 0;
  if ((block_dcmot[5].evout=calloc(block_dcmot[5].nevout,sizeof(double)))== NULL )return 0;
  if ((block_dcmot[5].outsz=malloc(sizeof(int)*block_dcmot[5].nout))== NULL ) return 0;
  if ((block_dcmot[5].outptr=malloc(sizeof(double*)*block_dcmot[5].nout))== NULL ) return 0;
  block_dcmot[5].outptr[0]=&(dcmot_block_outtb[1]);
  block_dcmot[5].outsz[0]=1;
  block_dcmot[5].z=&(z[5]);
  block_dcmot[5].rpar=&(RPAR1[18]);
  block_dcmot[5].work=(void **)(((double *)work)+5);
  block_dcmot[6].type = 0;
  block_dcmot[6].ztyp = 0;
  block_dcmot[6].ng = 0;
  block_dcmot[6].nz = 0;
  block_dcmot[6].nrpar = 0;
  block_dcmot[6].nipar = 1;
  block_dcmot[6].nin = 0;
  block_dcmot[6].nout = 0;
  block_dcmot[6].nevout = 1;
  block_dcmot[6].nmode = 0;
  if ((block_dcmot[6].insz=malloc(sizeof(int)*block_dcmot[6].nin))== NULL ) return 0;
  if ((block_dcmot[6].inptr=malloc(sizeof(double*)*block_dcmot[6].nin))== NULL ) return 0;
  if ((block_dcmot[6].evout=calloc(block_dcmot[6].nevout,sizeof(double)))== NULL )return 0;
  if ((block_dcmot[6].outsz=malloc(sizeof(int)*block_dcmot[6].nout))== NULL ) return 0;
  if ((block_dcmot[6].outptr=malloc(sizeof(double*)*block_dcmot[6].nout))== NULL ) return 0;
  block_dcmot[6].z=&(z[6]);
  block_dcmot[6].work=(void **)(((double *)work)+6);
  dcmot_init(block_dcmot,z,&t);
  return ;
}

int dcmot_rt_exec(scicos_block *block_dcmot,double *z, double *t)
{
int nevprt = 1;
    set_nevprt(nevprt);
    dcmotmain1(block_dcmot,z,t);
    dcmotmain2(block_dcmot,z,t);
}

/*---------------------------------------  Lapack messag function */
void
C2F(xerbla)(SRNAME,INFO,L)
     char *SRNAME;
     int *INFO;
     long int L;
{
  printf("** On entry to %s, parameter number %d  had an illegal value\n",SRNAME,*INFO);
}
/*--------------------------------------- main1 */
void
dcmotmain1(scicos_block *block_dcmot,double *z, double *t)
{
  dcmotddoit1(z, t, &(z[6]), (int *)(z+11));
```

```
}
/*---------------------------------------- main2 */
void
dcmotmain2(scicos_block *block_dcmot,double *z, double *t)
{
 dcmotddoit2(z, t, &(z[6]), (int *)(z+11));
}
/*---------------------------------------- init */
void
dcmot_init(scicos_block *block_dcmot,double *z, double *t)
{
 /*Block initializations*/
 dcmot_initi(t, &(z[6]), (int *)(z+11));
 /*Constants propagation*/
 dcmot_outtb(t, &(z[6]), (int *)(z+11));
}
/*---------------------------------------- end */
void
dcmot_end(scicos_block *block_dcmot,double *z, double *t)
{
 dcmot_endi(t, &(z[6]), (int *)(z+11));
}

/*---------------------------------------- ddoit1.c */
int
dcmotddoit1(z, told, outtb, iwa)

    double  *z, *told, *outtb;
    integer *iwa;
{
 /* System generated locals */

 /* Local variables */
 integer kiwa;

 /* Function Body */
 kiwa = 0;
 pointi=0+ totalnevprt;
 tevts[pointi-1]=*told;
 dcmotedoit1(told, &(z[6]), (int *)(z+11),&kiwa);
 iwa[1]=kiwa;
 return 0;

} /* ddoit1 */

/*---------------------------------------- edoit1.c */
int
dcmotedoit1(told, outtb, iwa, kiwa)
    double  *told, *outtb;
    integer *iwa, *kiwa;
{
 /* System generated locals */
 integer i2;

 /* Local variables */
 integer flag, kever, nport, nord, ierr1, ntvecm, nevprt;
```

```
double  rdouttb[6], *args[100];

/* Function Body */
kever = pointi;
pointi = evtspt[kever-1];
evtspt[kever-1] = -1;

nord = ordptr[kever] - ordptr[kever-1];
if (nord == 0) {
  return 0;
}
++(*kiwa);
iwa[*kiwa-1] = kever;
switch(kever) {
case 1:
  flag = 1 ;
  nevprt=1;
   block_dcmot[3].nevprt=nevprt;
  args[0]=&(outtb[3]);
  args[1]=&(outtb[0]);
  C2F(dsslti)(&flag,&block_dcmot[3].nevprt,told,block_dcmot[3].xd,
    block_dcmot[3].x,&block_dcmot[3].nx,block_dcmot[3].z,
    &block_dcmot[3].nz,block_dcmot[3].evout,&block_dcmot[3].nevout,
    block_dcmot[3].rpar,&block_dcmot[3].nrpar,block_dcmot[3].ipar,
    &block_dcmot[3].nipar,(double *)args[0],&nrd_1,(double *)args[1],
    &nrd_1);

     if(flag < 0 ) return(5 - flag);


  flag = 1 ;
  nevprt=1;
   block_dcmot[5].nevprt=nevprt;
  i_square_14(&block_dcmot[5], flag);

     if(flag < 0 ) return(5 - flag);


  flag = 1 ;
  nevprt=3;
   block_dcmot[4].nevprt=nevprt;
  summation(&block_dcmot[4], flag);

     if(flag < 0 ) return(5 - flag);


  flag = 1 ;
  nevprt=1;
   block_dcmot[2].nevprt=nevprt;
  args[0]=&(outtb[4]);
  args[1]=&(outtb[2]);
  C2F(dsslti)(&flag,&block_dcmot[2].nevprt,told,block_dcmot[2].xd,
    block_dcmot[2].x,&block_dcmot[2].nx,block_dcmot[2].z,
    &block_dcmot[2].nz,block_dcmot[2].evout,&block_dcmot[2].nevout,
    block_dcmot[2].rpar,&block_dcmot[2].nrpar,block_dcmot[2].ipar,
    &block_dcmot[2].nipar,(double *)args[0],&nrd_1,(double *)args[1],
```

```
     &nrd_1);

      if(flag < 0 ) return(5 - flag);

   flag = 1 ;
   nevprt=1;
    block_dcmot[1].nevprt=nevprt;
   args[0]=&(outtb[2]);
   args[1]=&(outtb[3]);
   C2F(samphold)(&flag,&block_dcmot[1].nevprt,told,block_dcmot[1].xd,
     block_dcmot[1].x,&block_dcmot[1].nx,block_dcmot[1].z,
     &block_dcmot[1].nz,block_dcmot[1].evout,&block_dcmot[1].nevout,
     block_dcmot[1].rpar,&block_dcmot[1].nrpar,block_dcmot[1].ipar,
     &block_dcmot[1].nipar,(double *)args[0],&nrd_1,(double *)args[1],
     &nrd_1);

      if(flag < 0 ) return(5 - flag);


   break;
  }
 return 0;
} /* edoit1 */

/*---------------------------------------- odoit.c */
int
dcmotodoit(told, outtb, iwa, phase)

    double  *told, *outtb;
    integer *iwa, phase;
{
 return 0;

} /* odoit */


/*---------------------------------------- ddoit2 */
int
dcmotddoit2(z, told, outtb, iwa)

    double  *z, *told, *outtb;
    integer *iwa;
{
 /* System generated locals */
 integer  i1, i;

 /* Local variables */
 integer flag, keve, kiwa, nport;
 double  *args[100];
 double  rdouttb[6];
 integer nevprt;

 /* Function Body */

 /*update continuous and discrete states on event */
 kiwa = iwa[1];
```

```
if (kiwa == 0) {
  return 0 ;
}
i1 = kiwa;
for (i = 1; i <= i1; ++i) {
  keve = iwa[i-1];
  switch(keve) {
  case 1:
  if (block_dcmot[3].nx+block_dcmot[3].nz > 0||
      *block_dcmot[3].work !=NULL) {
   flag = 2;
   nevprt=1;
   block_dcmot[3].nevprt=nevprt;
  args[0]=&(outtb[3]);
  args[1]=&(outtb[0]);
  C2F(dsslti)(&flag,&block_dcmot[3].nevprt,told,block_dcmot[3].xd,
    block_dcmot[3].x,&block_dcmot[3].nx,block_dcmot[3].z,
    &block_dcmot[3].nz,block_dcmot[3].evout,&block_dcmot[3].nevout,
    block_dcmot[3].rpar,&block_dcmot[3].nrpar,block_dcmot[3].ipar,
    &block_dcmot[3].nipar,(double *)args[0],&nrd_1,(double *)args[1],
    &nrd_1);

     if(flag < 0 ) return(5 - flag);
  }
  if (block_dcmot[5].nx+block_dcmot[5].nz > 0||
      *block_dcmot[5].work !=NULL) {
   flag = 2;
   nevprt=1;
   block_dcmot[5].nevprt=nevprt;
  i_square_14(&block_dcmot[5], flag);

     if(flag < 0 ) return(5 - flag);
  }
  if (block_dcmot[0].nx+block_dcmot[0].nz > 0||
      *block_dcmot[0].work !=NULL) {
   flag = 2;
   nevprt=1;
   block_dcmot[0].nevprt=nevprt;
  o_scope_SCOPE(&block_dcmot[0], flag);

     if(flag < 0 ) return(5 - flag);
  }
  if (block_dcmot[4].nx+block_dcmot[4].nz > 0||
      *block_dcmot[4].work !=NULL) {
   flag = 2;
   nevprt=3;
   block_dcmot[4].nevprt=nevprt;
  summation(&block_dcmot[4], flag);

     if(flag < 0 ) return(5 - flag);
  }
  if (block_dcmot[2].nx+block_dcmot[2].nz > 0||
      *block_dcmot[2].work !=NULL) {
   flag = 2;
   nevprt=1;
   block_dcmot[2].nevprt=nevprt;
```

```
    args[0]=&(outtb[4]);
    args[1]=&(outtb[2]);
    C2F(dsslti)(&flag,&block_dcmot[2].nevprt,told,block_dcmot[2].xd,
      block_dcmot[2].x,&block_dcmot[2].nx,block_dcmot[2].z,
      &block_dcmot[2].nz,block_dcmot[2].evout,&block_dcmot[2].nevout,
      block_dcmot[2].rpar,&block_dcmot[2].nrpar,block_dcmot[2].ipar,
      &block_dcmot[2].nipar,(double *)args[0],&nrd_1,(double *)args[1],
      &nrd_1);

       if(flag < 0 ) return(5 - flag);
    }
   if (block_dcmot[1].nx+block_dcmot[1].nz > 0||
       *block_dcmot[1].work !=NULL) {
     flag = 2;
     nevprt=1;
     block_dcmot[1].nevprt=nevprt;
    args[0]=&(outtb[2]);
    args[1]=&(outtb[3]);
    C2F(samphold)(&flag,&block_dcmot[1].nevprt,told,block_dcmot[1].xd,
      block_dcmot[1].x,&block_dcmot[1].nx,block_dcmot[1].z,
      &block_dcmot[1].nz,block_dcmot[1].evout,&block_dcmot[1].nevout,
      block_dcmot[1].rpar,&block_dcmot[1].nrpar,block_dcmot[1].ipar,
      &block_dcmot[1].nipar,(double *)args[0],&nrd_1,(double *)args[1],
      &nrd_1);

       if(flag < 0 ) return(5 - flag);
    }
   break;
   }
  }
  return 0;
} /* ddoit2 */
/*---------------------------------------- outtbini */
int
dcmot_outtb(told, outtb, iwa)

    /*Constants propagation*/
    double  *told, *outtb;
    integer *iwa;
{

  /* Local variables */
  integer flag;
  double  rdouttb[6];
  double  *args[100];
  integer nport;
  integer nevprt=0;

  /* Function Body */

  flag=1 ;
  return 0;
} /* dcmot_outtb */
/*---------------------------------------- initi */
int
dcmot_initi(told,outtb, iwa)
```

```
    /*Block initialization (flag=4)*/
    double  *told, *outtb;
    integer *iwa;
{

 /* Local variables */
 integer flag;
 double  rdouttb[6];
 double  *args[100];
 integer nport;
 integer nevprt=0;

 /* Function Body */

 flag=4 ;
  block_dcmot[0].nevprt=nevprt;
 o_scope_SCOPE(&block_dcmot[0], flag);

   if(flag < 0 ) return(5 - flag);
  block_dcmot[1].nevprt=nevprt;
 args[0]=&(outtb[2]);
 args[1]=&(outtb[3]);
 C2F(samphold)(&flag,&block_dcmot[1].nevprt,told,block_dcmot[1].xd,
   block_dcmot[1].x,&block_dcmot[1].nx,block_dcmot[1].z,
   &block_dcmot[1].nz,block_dcmot[1].evout,&block_dcmot[1].nevout,
   block_dcmot[1].rpar,&block_dcmot[1].nrpar,block_dcmot[1].ipar,
   &block_dcmot[1].nipar,(double *)args[0],&nrd_1,(double *)args[1],
   &nrd_1);

   if(flag < 0 ) return(5 - flag);
  block_dcmot[2].nevprt=nevprt;
 args[0]=&(outtb[4]);
 args[1]=&(outtb[2]);
 C2F(dsslti)(&flag,&block_dcmot[2].nevprt,told,block_dcmot[2].xd,
   block_dcmot[2].x,&block_dcmot[2].nx,block_dcmot[2].z,
   &block_dcmot[2].nz,block_dcmot[2].evout,&block_dcmot[2].nevout,
   block_dcmot[2].rpar,&block_dcmot[2].nrpar,block_dcmot[2].ipar,
   &block_dcmot[2].nipar,(double *)args[0],&nrd_1,(double *)args[1],
   &nrd_1);

   if(flag < 0 ) return(5 - flag);
  block_dcmot[3].nevprt=nevprt;
 args[0]=&(outtb[3]);
 args[1]=&(outtb[0]);
 C2F(dsslti)(&flag,&block_dcmot[3].nevprt,told,block_dcmot[3].xd,
   block_dcmot[3].x,&block_dcmot[3].nx,block_dcmot[3].z,
   &block_dcmot[3].nz,block_dcmot[3].evout,&block_dcmot[3].nevout,
   block_dcmot[3].rpar,&block_dcmot[3].nrpar,block_dcmot[3].ipar,
   &block_dcmot[3].nipar,(double *)args[0],&nrd_1,(double *)args[1],
   &nrd_1);

   if(flag < 0 ) return(5 - flag);
  block_dcmot[4].nevprt=nevprt;
 summation(&block_dcmot[4], flag);
```

```
    if(flag < 0 ) return(5 - flag);
  block_dcmot[5].nevprt=nevprt;
 i_square_14(&block_dcmot[5], flag);

    if(flag < 0 ) return(5 - flag);

 return 0;
} /* dcmot_initi */
/*---------------------------------------- endi */
/* file_end.c */
/* Subroutine */ int
dcmot_endi(told, outtb, iwa)

    double  *told, *outtb;
    integer *iwa;
{
 /* Local variables */
 integer flag;
 double  rdouttb[6];
 double  *args[100];
 integer nport;
 integer nevprt=0;

 /* Function Body */

 /*     ending  subroutine */
 flag=5 ;

  block_dcmot[0].nevprt=nevprt;
 o_scope_SCOPE(&block_dcmot[0], flag);

    if(flag < 0 ) return(5 - flag);

  block_dcmot[1].nevprt=nevprt;
 args[0]=&(outtb[2]);
 args[1]=&(outtb[3]);
 C2F(samphold)(&flag,&block_dcmot[1].nevprt,told,block_dcmot[1].xd,
   block_dcmot[1].x,&block_dcmot[1].nx,block_dcmot[1].z,
   &block_dcmot[1].nz,block_dcmot[1].evout,&block_dcmot[1].nevout,
   block_dcmot[1].rpar,&block_dcmot[1].nrpar,block_dcmot[1].ipar,
   &block_dcmot[1].nipar,(double *)args[0],&nrd_1,(double *)args[1],
   &nrd_1);

    if(flag < 0 ) return(5 - flag);

  block_dcmot[2].nevprt=nevprt;
 args[0]=&(outtb[4]);
 args[1]=&(outtb[2]);
 C2F(dsslti)(&flag,&block_dcmot[2].nevprt,told,block_dcmot[2].xd,
   block_dcmot[2].x,&block_dcmot[2].nx,block_dcmot[2].z,
   &block_dcmot[2].nz,block_dcmot[2].evout,&block_dcmot[2].nevout,
   block_dcmot[2].rpar,&block_dcmot[2].nrpar,block_dcmot[2].ipar,
   &block_dcmot[2].nipar,(double *)args[0],&nrd_1,(double *)args[1],
   &nrd_1);

    if(flag < 0 ) return(5 - flag);
```

```
  block_dcmot[3].nevprt=nevprt;
 args[0]=&(outtb[3]);
 args[1]=&(outtb[0]);
 C2F(dsslti)(&flag,&block_dcmot[3].nevprt,told,block_dcmot[3].xd,
   block_dcmot[3].x,&block_dcmot[3].nx,block_dcmot[3].z,
   &block_dcmot[3].nz,block_dcmot[3].evout,&block_dcmot[3].nevout,
   block_dcmot[3].rpar,&block_dcmot[3].nrpar,block_dcmot[3].ipar,
   &block_dcmot[3].nipar,(double *)args[0],&nrd_1,(double *)args[1],
   &nrd_1);

    if(flag < 0 ) return(5 - flag);

  block_dcmot[4].nevprt=nevprt;
 summation(&block_dcmot[4], flag);

    if(flag < 0 ) return(5 - flag);

  block_dcmot[5].nevprt=nevprt;
 i_square_14(&block_dcmot[5], flag);

    if(flag < 0 ) return(5 - flag);


 return 0;
} /* ending */

int dcmot_putevs(t, evtnb, ierr)
 double *t;
 integer *evtnb, *ierr;
{
 *ierr = 0;
 if (evtspt[*evtnb-1] != -1) {
   *ierr = 1;
   return 0;
 } else {
   evtspt[*evtnb-1] = 0;
   tevts[*evtnb-1] = *t;
 }
 if (pointi == 0) {
   pointi = *evtnb;
   return 0;
 }
 evtspt[*evtnb-1] = pointi;
 pointi = *evtnb;
 return 0;
}
void set_block_error(int err)
{
 return;
}
int get_phase_simulation()
{
 return 1;
}
void * scicos_malloc(size_t size)
```

```
{
  return malloc(size);
}
void scicos_free(void *p)
{
  free(p);
}
double get_scicos_time()
{
  return(TIME);
}
void do_cold_restart()
{
  return;
}
void sciprint (char *fmt){
  return;
}
```