Universidad de las Ciencias Informáticas

Facultad 10

"Empotramiento de Sistemas Operativos basados en GNU/Linux."

Trabajo de diploma para optar por el título de Ingeniero Informático

Autor (es): Yusdiel Rodríguez Amoros

René Raúl Sarzo Pavón

Tutor (es): Lic. Israel González Delgado

Ing. Tte. Rassiel Valiente Mesa

Clasificación: Investigativa

Clasificación del área de desarrollo: Asimilación Tecnológica, Técnica de Programación, Sistemas Operativos.

Curso Docente: 2008-2009

Ciudad de La Habana

Junio, 2009

Agradecimientos

A mis padres por darme una gran educación, por ser mis ejemplos a seguir y por todo el cariño y amor que me ofrecen cada día.

A mi hermana por todo el cariño y la ayuda brindada.

A mi novia Odesa James Canela por toda la paciencia, ayuda y amor que me dio en todo este tiempo.

A mis amigos que estuvieron siempre a mi lado cuando los necesite, especialmente a Yusdiel por estar en todo momento y haberme ayudado a cumplir este sueño.

René

A mis padres por haberme encaminado en la vida y haberme dado el amor y apoyo que necesitaba.

A mi padrastro Armando Orchet por haberme dado una gran educación.

A mi amigo René por haberme ayudado a cumplir este sueño.

Yusdiel

También queremos agradecer a nuestros tutores, por la gran ayuda brindada.

A personas importantes como Yosley, Vega, Mijail y el profesor Pedro Martinto.

Dedicatoria

A mis padres, mi hermana y a mi novia.

A todas las personas que me han ayudado.

René

A mis padres y a mi padrastro.

A todas aquellas personas son importantes para mí.

Yusdiel

Resumen

El siguiente trabajo describe el proceso de creación de un sistema embebido, para esto se describen dos tareas principales: la primera es la realización de una distribución basada en GNU/Linux, utilizando la herramienta de compilación cruzada Buildroot y un kernel versión 2.6.28.2, la otra tarea fue realizar la investigación, propuesta y compra del dispositivo a empotrar. Se espera como resultado, obtener una distribución propia que controle el dispositivo propuesto y que sirva como base para la posterior incorporación de aplicaciones dedicadas al encaminamiento de paquetes y el cifrado en las redes del MININT.

Índice

Introducción

El mundo actual se encuentra estrechamente conectado, lo que hace unos años parecía imposible de lograr, las comunicaciones es un sector que ha proporcionado grandes avances y cumple un papel importante en la actualidad.

El desarrollo de las redes y la necesidad de intercambio de información han tenido un inevitable crecimiento, hasta llegar a formar una red de redes con una interconexión universal. En la actualidad la red de redes de mayor alcance que existe se denomina Internet. Internet ha llegado a gran parte de los hogares y de las empresas de los países ricos, en este aspecto se ha abierto una brecha digital con los países pobres, en los cuales la penetración de Internet y las nuevas tecnologías es muy limitada para las personas. No obstante, en el transcurso del tiempo se ha venido extendiendo el acceso a Internet en casi todas las regiones del mundo, de modo que es relativamente sencillo encontrar por lo menos dos computadoras conectadas en regiones remotas.

Todo este desarrollo de las redes no hubiera sido posible sin los dispositivos de hardware especializados para interconectar ordenadores. Existen diversos dispositivos dependiendo del propósito de uso como son: los Switch, los Routers, los Modems y los Hub. Estos equipos tienen una gran importancia para la transmisión de los datos de una red, ya que son los encargados de controlar, gestionar y mantener el transporte de los paquetes que transitan por la misma. La administración del funcionamiento del hardware de estos dispositivos, se realiza a través de un sistema operativo. En la actualidad existen numerosos Sistemas Operativos con diferentes propósitos de uso. Entre los más populares se encuentran GNU/Linux, BSD, Solaris, MacOSX y Windows.

El MININT se encuentra desarrollando soluciones informáticas basadas en GNU/Linux como son: la distribución NovaRouter y el Sistema de Protección de Perimetral (SPP)[1]. La distribución NovaRouter tiene como propósito encaminar los paquetes que transitan por la red, por otra parte el SPP es el encargado de proteger los datos cuando pasan de una subred a otra.

La distribución NovaRouter y el SPP son soluciones concebidas para dispositivos de red, pero se encuentran en una computadora personal lo que no permite explotar al máximo sus potencialidades. Esta computadora cumple con la función de velar y asegurar toda la

[1] Sistema encargado de la protección de los datos que transitan por la red del MININT.

información que entra y sale de la institución. Esta situación provoca otros problemas, como son:

La **instalación** es uno de los procesos demasiado complejo para el usuario final (Administrador), este debe ser capacitado para esta tarea por lo que retrasa la explotación del producto. El estar hospedado en una computadora personal afecta el **rendimiento del sistema**, hay recursos de esta que no son necesarios como son: la salida de video, la impresora, teclado, lector de disquetes, lector de disco (CD) y mouse, eliminar esto traería un mayor aprovechamiento del hardware dedicado al encaminamiento de paquetes. El **costo** es un factor importante, una computadora personal tiene un costo que oscila entre los 500 y 1000 dólares, un dispositivo de red oscila entre 100 y 300 dólares, esto provoca un gasto innecesario. La **configuración** es un proceso que implica conocimientos de hardware, solamente no tiene que configurar opciones dedicadas al encaminamiento de paquetes sino debe configurar aspectos del sistema operativo, esto debería ser transparente al cliente. El sistema y las aplicaciones que corren no se encuentran **tuneados** por lo que es necesario afinar sus configuraciones. La **homogeneidad** es un aspecto que afecta bastante, el hardware donde está hospedado la distribución es demasiado heterogéneo por lo que implica conocimientos de aplicaciones adicionales como drivers y plugins, por estas razones el control de versiones y la corrección de errores redobla los esfuerzos por parte de los desarrolladores, ya que no existe una plataforma homogénea. La **seguridad** es fundamental, mientras más procesos estén activados se tiene más posibilidad de un ataque informático.

La presente investigación surge como necesidad de dar solución a las situaciones antes expuestas, por lo que se presenta el siguiente **problema a resolver**: ¿Cómo empotrar una distribución GNU/Linux en un dispositivo de red?

Como **objeto de estudio** se tiene el proceso de desarrollo de las distribuciones GNU/Linux en dispositivos de red pertenecientes al MININT.

Objetivo general, empotrar una distribución propia basada en GNU/Linux en un dispositivo autónomo, que sirva como base para la posterior incorporación de la distribución NovaRouter y el Sistema de Protección Perimetral.

Campo de acción se centra en el proceso de desarrollo de las distribuciones GNU/Linux empotradas en dispositivos de red en instituciones del MININT.

La **idea a defender** en este trabajo es: El encaminamiento de paquetes y el rendimiento en la red del MININT será posible con el empotramiento de la distribución GNU/Linux propia en un dispositivo de red, y servirá como base para la posterior incorporación del SPP y la distribución NovaRouter.

Para dar cumplimiento al objetivo general planteado se han definido una serie de **tareas de investigación:**

1. Analizar el funcionamiento de la distribución NovaRouter.
2. Investigar sobre el comportamiento de los kernel de las distribuciones libres.
3. Buscar, analizar y comparar dispositivos de red para empotrar Sistemas Operativos.
4. Investigar sobre herramientas para el empotramiento de Sistemas Operativos.
5. Investigar sobre sistemas embebidos existentes.
6. Instrumentar el empotramiento de la distribución propia.
7. Desarrollar pruebas en laboratorio de eficacia y eficiencia.

Los **métodos científicos** utilizados en el transcurso de la investigación son: el **Analítico-Sintético**, con el cual se realizó el estudio del funcionamiento de las soluciones desarrolladas por el MININT anteriormente mencionadas, el comportamiento de los kernel de las distribuciones libres, del proceso de encaminamiento de paquetes y de los equipamientos para el empotramiento de estos sistemas, el **Histórico-Lógico** para la investigación de las tendencias actuales y el estado del arte de las herramientas utilizadas para el empotramiento de Sistemas Operativos y de los sistemas embebidos existentes, la **Observación** y el **Experimento** se utilizaron en las pruebas de laboratorios con dispositivos que cumplían con los requisitos para este proceso, lo que nos sirvió para recoger información útil y sacar conclusiones importantes.

El presente trabajo está conformado por cuatro capítulos, en los cuales se abordará todo el estudio realizado, así como la descripción de la solución propuesta.

Capítulo 1: Contiene la **Fundamentación teórica** del trabajo, abordando todos los elementos

teóricos que sustentan el problema a resolver y el objetivo del trabajo, se describen los principales elementos de las distribuciones existentes en el mundo, además se explica el concepto y funcionamiento de los sistemas embebidos y se mencionan diversos dispositivos estudiados para realizar la propuesta final.

Capítulo 2: Se realiza la descripción del proceso de creación del entorno de compilación cruzada mediante la herramienta Buildroot.

Capítulo 3: Se detallan los principales pasos y herramientas utilizados para la creación de la distribución.

Capítulo 4: Se detallan las pruebas que se hicieron con el producto.

Capítulo

1

1-Fundamentación Teórica

1.1 Introducción

En este capítulo se exponen los principales elementos en los cuales se basó la realización de este trabajo de diploma. Aborda el significado de software libre, así como el concepto de GNU/Linux y la importancia de su utilización en este trabajo de diploma. Otros aspectos tratados son las distribuciones GNU/Linux y los diferentes tipos de dispositivos de red, además se incorpora un poco del estado actual y la definición de los sistemas empotrados o embebidos.

1.2 Software Libre

El software libre es una cuestión de libertad, no de precio. Para entender el concepto, se debería pensar en libre como en libre expresión, no como en barra libre.

El software libre es una cuestión de la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Más precisamente, se refiere a cuatro tipos de libertades para los usuarios del software:

- La libertad de ejecutar el programa, para cualquier propósito (libertad 0).
- La libertad de estudiar cómo trabaja el programa, y adaptarlo a sus necesidades (libertad 1).
- La libertad de redistribuir copias para que pueda ayudar al prójimo (libertad 2).
- La libertad de mejorar el programa y publicar sus mejoras, y versiones modificadas en general, para que se beneficie toda la comunidad (libertad 3).

Un programa es software libre si los usuarios tienen todas esas libertades. Entonces, debería ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratis o cobrando una

tarifa por distribución, a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir o pagar el permiso.

También debería tener la libertad de hacer modificaciones y usarlas en privado, en su propio trabajo u obra, sin siquiera mencionar que existen. Si publica sus cambios, no debería estar obligado a notificarlo a alguien en particular, o de alguna forma en particular.

La libertad de ejecutar el programa significa la libertad para cualquier tipo de persona u organización de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y propósito, sin estar obligado a comunicarlo a su programador, o a alguna otra entidad específica. En esta libertad, el propósito de los *usuarios* es el que importa, no el propósito de los *programadores*. Como usuario es libre de ejecutar un programa para sus propósitos; y si lo distribuye a otra persona, también es libre para ejecutarlo para sus propósitos, pero usted no tiene derecho a imponerle sus propios propósitos.

La libertad de redistribuir copias debe incluir las formas binarias o ejecutables del programa, así como el código fuente, tanto para las versiones modificadas como para las no lo están. (Distribuir programas en forma de ejecutables es necesario para que los Sistemas Operativos libres se puedan instalar fácilmente). Resulta aceptable si no existe un modo de producir una formato binario o ejecutable para un programa específico, dado que algunos lenguajes no incorporan esa característica, pero debe tener la libertad de redistribuir dichos formatos si encontrara o programara una forma de hacerlo.

Para que las libertades para realizar cambios y publicar versiones mejoradas, tengan sentido, debe tener acceso al código fuente del programa. Por consiguiente, el acceso al código fuente es una condición necesaria para el software libre.

Una manera importante de modificar un programa es fusionando subrutinas y módulos libres disponibles. Si la licencia del programa dice que no puede fusionar un módulo existente con una debida licencia, así como si le requiere ser el titular de los derechos de autor de lo que agregue, entonces la licencia es demasiado restrictiva para calificarla como libre.

Para que estas libertades puedan ser reales, deben ser irrevocables siempre que usted no cometa ninguna equivocación; si el programador del software tiene el poder de revocar la licencia, o de cambiar retroactivamente sus términos, sin que usted se haya equivocado para justificarlo, el software no es libre.

Tampoco debe confundirse software libre con "software de dominio público". Éste último es aquel que no requiere de licencia, pues sus derechos de explotación son para toda la humanidad, porque pertenece a todos por igual. Cualquiera puede hacer uso de él, siempre con fines legales y consignando su autoría original. Este software sería aquel cuyo autor lo dona a la humanidad o cuyos derechos de autor han expirado, tras un plazo contado desde la muerte de éste, habitualmente 70 años. Si un autor condiciona su uso bajo una licencia, por muy débil que sea, ya no es dominio público.

1.3 GNU/Linux

Debido a un particular giro de acontecimientos, la versión de GNU más ampliamente usada hoy es, con frecuencia, más conocida como «Linux», y muchos usuarios no son conscientes del alcance de su conexión con el Proyecto GNU.

Efectivamente hay un Linux, y las personas lo usan, pero no es el sistema operativo. Linux es el núcleo: el programa del sistema que asigna los recursos de la máquina a los otros programas que usted ejecute. El núcleo es una parte esencial de todo sistema operativo, pero inútil por sí solo; solo puede funcionar en el contexto de un sistema operativo completo. Linux se usa normalmente en combinación con el sistema operativo GNU: el sistema completo es básicamente GNU, con Linux actuando de núcleo.

El Proyecto GNU apoya tanto a los sistemas GNU/Linux como al sistema GNU. Financia la reescritura de las extensiones relacionadas con Linux de la biblioteca de C de GNU, de modo que ahora se integran bien, y los sistemas GNU/Linux modernos usan la versión actual de la biblioteca sin necesidad de hacerle modificaciones. También financió las primeras etapas del desarrollo de Debian GNU/Linux.

GNU/Linux es un Sistema Operativo de software libre que cumple las normas POSIX, su base es un núcleo o Kernel monolítico llamado Linux combinado con un grupo de librerías y herramientas. Su estructura general es la típica de cualquier sistema UNIX (núcleo, "intérprete de comandos", aplicaciones). GNU/Linux tiene todas las características que se pueden esperar de un moderno y flexible Sistema Operativo. Incluye multitarea real, memoria virtual, librerías compartidas, dirección y manejo propio de memoria. Es sin lugar a dudas uno de los ejemplos más prominentes del software libre y del desarrollo del código abierto.

Actualmente se usó sistemas GNU basados en Linux para la mayor parte del trabajo, y se espera que usted también lo haga. Pero por favor, no confunda a la gente usando el nombre Linux de manera ambigua. Linux es el núcleo, uno de los principales componentes del sistema. El sistema en su conjunto es más o menos el sistema GNU, con Linux añadido. Cuando hable de esta combinación, por favor, llámela GNU/Linux. **(3)**

1.4 Importancia de la utilización del GNU/Linux en el trabajo de diploma.

- Multitarea: La palabra multitarea describe la habilidad de ejecutar varios programas al mismo tiempo. GNU/Linux utiliza la llamada multitarea preventiva, la cual asegura que todos los programas que se están utilizando en un momento dado serán ejecutados, siendo el Sistema Operativo el encargado de ceder tiempo de microprocesador a cada programa.
- Multiusuario: Varios usuarios usando la misma máquina al mismo tiempo. Multiplataforma: Las plataformas en las que en un principio se puede utilizar Linux son 386-, 486-. Pentium, Pentium Pro, Pentium II, Amiga y Atari, también existen versiones para su utilización en otras plataformas, como Alpha, ARM, MIPS (Microprocessor without Interlocked Pipeline Stages), PowerPC y SPARC. Multiprocesador: Soporte para sistemas con más de un procesador, está disponible para Intel y SPARC.
- Funciona en modo protegido 386.
- Protección de la memoria entre procesos, de manera que uno de ellos no pueda “colgar” (interrumpir, colapsar) el sistema.
- Carga de ejecutables por demanda: GNU/Linux sólo lee del disco aquellas partes de un programa que están siendo usadas actualmente.
- Política de copia en escritura para compartir páginas entre ejecutables: esto significa que varios procesos pueden usar la misma zona de memoria para ejecutarse. Cuando alguno intenta escribir en esa memoria, la página (4Kb de memoria) se copia a otro lugar. Esta política de copia en escritura tiene dos beneficios: aumenta la velocidad y reduce el uso de memoria.
- Memoria virtual usando paginación (sin intercambio de procesos completos) a disco: a una partición o un archivo en el sistema de archivos, o ambos, con la posibilidad de

añadir más áreas de intercambio sobre la marcha. Un total de 16 zonas de intercambio de 128Mb de tamaño máximo pueden ser usadas en un momento dado con un límite teórico de 2Gb para intercambio. Este límite se puede aumentar fácilmente con el cambio de unas cuantas líneas en el código fuente.

- Compatible con POSIX, System V y BSD a nivel fuente.
- Todo el código fuente está disponible, incluyendo el núcleo completo y todos los drivers, las herramientas de desarrollo y todos los programas de usuario; además todo ello se puede distribuir libremente. Hay algunos programas comerciales que están siendo ofrecidos para GNU/Linux actualmente sin código fuente, pero todo lo que ha sido gratuito sigue siendo gratuito.
- Control de tareas POSIX.
- Emulación de 387 en el núcleo, de tal forma que los programas no tengan que hacer su propia emulación matemática. Cualquier máquina que ejecute GNU/Linux parecerá dotada de coprocesador matemático. Por supuesto, si el ordenador ya tiene una FPU (unidad de punto flotante), esta será usada en lugar de la emulación, pudiendo incluso compilar tu propio Kernel sin la emulación matemática y conseguir un pequeño ahorro de memoria.
- Soporte para muchos teclados nacionales o adaptados y es bastante fácil añadir nuevos dinámicamente.
- Consolas virtuales múltiples: varias sesiones de login a través de la consola entre las que se puede cambiar con las combinaciones adecuadas de teclas (totalmente independiente del hardware de video). Se crean dinámicamente y puedes tener hasta 64.
- Soporte para varios sistemas de archivo comunes, incluyendo minix-1, Xenix y todos los sistemas de archivo típicos de System V, y tiene un avanzado sistema de archivos propio con una capacidad de hasta 4 Tb y nombres de archivos de hasta 255 caracteres de longitud.
- Acceso transparente a particiones MS-DOS (o a particiones OS/2 FAT) mediante un sistema de archivos especial: no es necesario ningún comando especial para usar la partición MS-DOS, esta parece un sistema de archivos normal de Unix (excepto por algunas restricciones en los nombres de archivo, permisos, etc.). Las particiones comprimidas de MS-DOS no son accesibles en este momento, y no se espera que lo sean en el futuro. El soporte para VFAT, FAT32 (WNT, Windows 95/98) se encuentra

soportado desde la versión 2.0 del núcleo y el NTFS de WNT desde la versión 2.2 (Este último solo en modo lectura).

- Sistema de archivos de CD-ROM que lee todos los formatos estándar de CD-ROM.
- TCP/IP, incluyendo ftp, telnet, NFS, etc.
- Lan Manager / Windows Native (SMB), software cliente y servidor.
- Diversos protocolos de red incluidos en el kernel: TCP, IPv4, IPv6, AX.25, X.25, IPX, DDP, Netrom, etc. **(8)**

1.5 Distribución

Se le denomina distribución (distros) a las diferentes variantes que existen del Sistema Operativo GNU/Linux, estas incorporan un conjunto de aplicaciones y determinados paquetes con el objetivo de satisfacer a un grupo específico de usuarios. Una distribución es una compilación de aplicaciones que junto al núcleo o Kernel de Linux se empaqueta y configura, de manera tal que permita su fácil y rápida instalación. Cada distribución puede incluir cualquier número de software adicional como se estime conveniente.

1.5.1 Ejemplos de Distribuciones

Debian GNU/Linux es la principal distribución Linux del proyecto **Debian**, que basa su principio y fin en el software libre. Creada por el proyecto Debian en el año 1993, la organización responsable de la creación y mantenimiento de la misma distribución, centrado en el núcleo de Linux y utilidades GNU. Éste también mantiene y desarrolla sistemas GNU basados en otros núcleos (Debian GNU/Hurd, Debian GNU/NetBSD y Debian GNU/kFreeBSD).

Nace como una apuesta por separar en sus versiones el software libre del software no libre. El modelo de desarrollo es independiente a empresas, creado por los propios usuarios, sin depender de ninguna manera de necesidades comerciales. Debian no vende directamente su software, lo pone a disposición de cualquiera en Internet, aunque sí permite a personas o empresas distribuir comercialmente este software mientras se respete su licencia.

Ubuntu es una distribución GNU/Linux que ofrece un sistema operativo predominantemente enfocado a computadores personales, aunque también proporciona soporte para servidores. Es una de las más importantes distribuciones de GNU/Linux a nivel mundial. Se basa en Debian

GNU/Linux y concentra su objetivo en la facilidad y libertad de uso, la fluida instalación y los lanzamientos regulares (cada 6 meses: las versiones .04 en abril y las .10 en octubre). El principal patrocinador es Canonical Ltd., una empresa privada fundada y financiada por el empresario sudafricano Mark Shuttleworth. El nombre de la distribución proviene del concepto zulú y xhosa de Ubuntu, que significa "humanidad hacia otros" o "yo soy porque nosotros somos". Ubuntu es un movimiento sudafricano encabezado por el obispo Desmond Tutu, quien ganó el Premio Nobel de la Paz en 1984 por sus luchas en contra del Apartheid en Sudáfrica. El sudafricano Mark Shuttleworth, mecenas del proyecto, se encontraba muy familiarizado con la corriente. Tras ver similitudes entre los ideales de los proyectos GNU, Debian y en general con el movimiento del software libre, decidió aprovechar la ocasión para difundir los ideales de Ubuntu. El eslogan de la distribución –"Linux para seres humanos" (en inglés "Linux for Human Beings") – resume una de sus metas principales: hacer de GNU/Linux un sistema operativo más accesible y fácil de usar. **(6)**

SUSE Linux es una de las más conocidas distribuciones Linux existentes a nivel mundial, se basó en sus orígenes en Slackware. Entre las principales virtudes de esta distribución se encuentra el que sea una de las más sencillas de instalar y administrar, ya que cuenta con varios asistentes gráficos para completar diversas tareas en especial por su gran herramienta de instalación y configuración YasT.

Su nombre "SuSE" es el acrónimo, en alemán *"*Software und Systementwicklung*"* (*Desarrollo de Sistemas y de Software*), el cual formaba parte del nombre original de la compañía y que se podría traducir como "desarrollo de software y sistemas". El nombre actual de la compañía es *SuSE LINUX*, habiendo perdido el primer término su significado (al menos oficialmente).

El 4 de noviembre de 2003, la compañía multinacional estadounidense Novell anunció que iba a comprar *SuSE LINUX*. La adquisición se llevó a cabo en enero de 2004. En el año 2005, en la LinuxWorld, Novell, siguiendo los pasos de RedHat Inc., anunció la liberación de la distribución *SuSE Linux* para que la comunidad fuera la encargada del desarrollo de esta distribución, que ahora se denomina openSUSE.

NovaRouter es una distribución dedicada al encaminamiento de paquetes, fue el trabajo de diploma de José Antonio Vega Hermosilla y Yosley Bello Rodriguez. Se basa en Nova, distribución realizada por un grupo de estudiantes y profesores de la Universidad de las Ciencias Informáticas, la cual tiene sus cimientos en Gentoo y surge a mediados del 2005

como respuesta a la necesidad de una plataforma que garantice la compatibilidad de las aplicaciones que están en desarrollo con sistemas libres. NovaRouter se elabora para dar una solución sobre GNU/Linux orientada al enrutamiento de paquetes TCP/IP, junto a la posterior adición a esta de otros módulos especializados en la protección de datos por parte del MININT, dicha solución garantizaría la seguridad e integridad respectivamente de las redes informáticas con las que cuenta el MININT.

1.6 Dispositivos de Red

Es el hardware que permite comunicar las computadoras que existen en una red, realizar la conexión con un Proveedor de Servicios de Internet, controlar el tráfico de información que transita por una red e interconectar redes heterogéneas.

1.6.1 Ejemplos de dispositivos de Red

NIC/MAU (Tarjeta de red)

"Network Interface Card" (Tarjeta de interfaz de red) o "Medium Access Unit" (Medio de unidad de acceso). Cada computadora necesita el "hardware" para transmitir y recibir información. Es el dispositivo que conecta la computadora u otro equipo de red con el medio físico. La NIC es un tipo de tarjeta de expansión de la computadora y proporciona un puerto en la parte trasera de la PC al cual se conecta el cable de la red. Hoy en día cada vez son más los equipos que disponen de interfaz de red, principalmente Ethernet, incorporadas. A veces, es necesario, además de la tarjeta de red, un transceptor. Este es un dispositivo que se conecta al medio físico y a la tarjeta, bien porque no sea posible la conexión directa (10 base 5) o porque el medio sea distinto del que utiliza la tarjeta.

Hubs (Concentradores)

Son equipos que permiten estructurar el cableado de las redes. La variedad de tipos y características de estos equipos es muy grande. En un principio eran solo concentradores de cableado, pero cada vez disponen de mayor número de capacidad de la red, gestión remota, etc. La tendencia es a incorporar más funciones en el concentrador. Existen concentradores para todo tipo de medios físicos.

Repetidores

Son equipos que actúan a nivel físico. Prolongan la longitud de la red uniendo dos segmentos y amplificando la señal, pero junto con ella amplifican también el ruido. La red sigue siendo una sola, con lo cual, siguen siendo válidas las limitaciones en cuanto al número de estaciones que pueden compartir el medio.

Bridges (Puentes)

Son equipos que unen dos redes actuando sobre los protocolos de bajo nivel, en el nivel de control de acceso al medio. Sólo el tráfico de una red que va dirigido a la otra atraviesa el dispositivo. Esto permite a los administradores dividir las redes en segmentos lógicos, descargando de tráfico las interconexiones. Los bridges producen las señales, con lo cual no se transmite ruido a través de ellos.

Routers (Encaminadores)

Son equipos de interconexión de redes que actúan a nivel de los protocolos de red. Permiten utilizar varios sistemas de interconexión mejorando el rendimiento de la transmisión entre redes. Su funcionamiento es más lento que el de los bridges pero su capacidad es mayor. Permiten, incluso, enlazar dos redes basadas en un protocolo, por medio de otra que utilice un protocolo diferente.

Gateways

Son equipos para interconectar redes con protocolos y arquitecturas completamente diferentes a todos los niveles de comunicación. La traducción de las unidades de información reduce mucho la velocidad de transmisión a través de estos equipos.

Servidores

Son equipos que permiten la conexión a la red de equipos periféricos tanto para la entrada como para la salida de datos. Estos dispositivos se ofrecen en la red como recursos compartidos. Así un terminal conectado a uno de estos dispositivos puede establecer sesiones

contra varios ordenadores multiusuario disponibles en la red. Igualmente, cualquier sistema de la red puede imprimir en las impresoras conectadas a un servidor.

Módems

Son equipos que permiten a las computadoras comunicarse entre sí a través de líneas telefónicas; modulación y desmodulación de señales electrónicas que pueden ser procesadas por computadoras. Los módems pueden ser externos (un dispositivo de comunicación) o internos (dispositivo de comunicación interno o tarjeta de circuitos que se inserta en una de las ranuras de expansión de la computadora).

1.6.2 Dispositivos Consultados

RouterBoard 230

RouterBoard 532

VIA NAB 7500

1.7 Sistemas Empotrados o Embebidos

1.7.1 ¿Qué es un sistema embebido?

Cuando se habla de un sistema embebido se está haciendo referencia a un dispositivo que a diferencia de una computadora (PC) de propósito general, es de uso único. Está diseñado para realizar un conjunto específico de tareas u operaciones. Por lo general, son computadoras de una única placa o tarjeta, denominados SBC (del inglés, Single Board Computer). Estos dispositivos funcionan con un sistema operativo almacenado en memorias internas de la misma placa y en la mayoría de los casos son desarrollados por los mismos fabricantes de la SBC. Ejemplos de dispositivos que poseen sistemas embebidos son routers, switchs, firewalls, entre otros.

1.7.2 Ventajas de un sistema empotrado sobre las soluciones industriales tradicionales

- Posibilidad de utilización de Sistemas Operativos potentes que ya realizan numerosas tareas: comunicaciones por redes de datos, soporte gráfico, concurrencia con lanzamiento de threads[2], etc. Estos Sistemas Operativos pueden ser los mismos que para PC compatibles (Linux, Windows, MS-DOS) con fuertes exigencias en hardware o bien ser una versión reducida de los mismos con características orientadas a los PC empotrados.
- Al utilizar dichos Sistemas Operativos se pueden encontrar fácilmente herramientas de desarrollo de software potentes así como numerosos programadores que las dominan, dada la extensión mundial de las aplicaciones para PC compatibles.
- Reducción en el precio de los componentes de hardware y software debido a la gran cantidad de PCs en el mundo.

1.8 Sistemas GNU/Linux Embebidos

1.8.1 ¿En qué consiste el proceso de implementación de un sistema GNU/Linux embebido?

Para implementar un GNU/Linux embebido en un dispositivo de hardware, se debe conocer en términos generales la arquitectura del mismo, es decir, el tipo de micro-procesador que posee, la cantidad de memoria, los buses que soporta, los componentes que posee la placa, etc. Esta información es de vital importancia, ya que al preparar el sistema GNU/Linux que se ejecutará en el dispositivo, se le debe compilar con soporte para esas características. La tarea de construir un GNU/Linux que se ejecute en este dispositivo objetivo (target) se debe realizar mediante compilación cruzada. Por esto, es imprescindible crear un entorno de compilación cruzada en una computadora de escritorio (host), bajo este entorno se realizará la compilación del kernel, del sistema de archivos y de las aplicaciones que luego serán ejecutadas en el dispositivo. Por último se debe realizar el traspaso del sistema GNU/Linux al equipo objetivo, tres posibles maneras de transferir el sistema es utilizando un medio externo como una memoria compact flash, un disco con conector IDE o mediante la red.

---

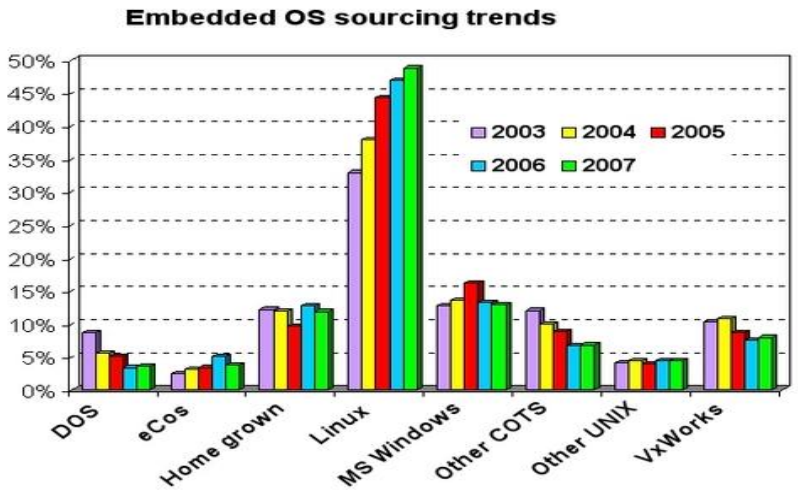
[2] Un hilo de ejecución, en Sistemas Operativos, es una característica que permite a una aplicación realizar varias tareas concurrentemente.

1.8.2 Antecedentes de sistemas GNU/Linux embebido

El sistema operativo GNU/Linux, tal como se conoce, realiza su primera aparición en 1991 cuando se le suma al proyecto creado por Richard Stallman en 1983, denominado GNU (acrónimo recursivo que significa GNU is Not Unix) el kernel desarrollado por Linus Torvals. Luego de que Richard Stallman crea la Fundación de Software Libre (FSF, del inglés Free Software Foundation) se da origen a la Licencia Pública General GNU (GPL, del inglés GNU General Public Licence) posibilitando el desarrollo de software libre en el sistema de copyright. Esta licencia permitía a cualquier persona, programador o no, acceder al código fuente para realizar aportes y modificaciones libremente, manteniendo este software modificado bajo la misma licencia. Este modelo de desarrollo de código abierto fomenta una comunidad que se retroalimenta con la colaboración, a través de Internet, de todos los programadores alrededor del mundo. Este es el principal motivo que justifica el continuo crecimiento y expansión de GNU/Linux.

Si bien GNU/Linux ha sido desarrollado para funcionar en servidores y computadoras de escritorio, en los últimos años y debido a su código abierto se ha ido diversificando su campo de aplicación

Así fue como en 1996 surgió un proyecto de investigación denominado *RT-Linux* para implementar GNU/Linux en sistemas de tiempo real, utilizando un kernel pequeño y compacto. Un año más tarde, el proyecto *uClinux*, comenzaba con el objetivo de utilizar GNU/Linux en procesadores sin unidades de administración de memoria (MMU, del inglés Memory Management Unit), es a partir de esta fecha y durante los 6 años subsiguientes que se comienza a utilizar ampliamente GNU/Linux en sistemas embebidos.



A continuación se realiza una breve descripción de los acontecimientos más relevantes relacionados con los sistemas embebidos basados en GNU/Linux:

Año 1999

- En la Conferencia de Sistemas Embebidos (ESC, del inglés Embedded Systems Conference) de Septiembre de 1999 compañías como Lineo, FSM Labs, MontaVista y Zentropix anuncian su soporte a sistemas Linux embebidos.
- Rick Lehrbaum comienza el portal de Linux embebido: Linuxdevices.com.
- BlueCat Linux fue anunciada como la primera distribución comercial de Linux embebido.

Año 2000

Una gran cantidad de compañías adoptan Linux embebido en sus líneas de productos

- Samsung lanza Yopy, una PDA con Linux embebido.
- Ericsson lanza el HS210, un teléfono con pantalla que combina conectividad inalámbrica con acceso a internet, telefonía y funciones de correo electrónico.
- Atmel anuncia el lanzamiento de un dispositivo de un único chip basado en Linux, el AT75C310, que incluye soporte para VoIP y audio.

El desarrollo de herramientas y utilitarios para utilizar en el desarrollo de sistemas Linux embebidos

- Busybox
- Goahead
- Trolltech
- ViewML

Otro hecho más que importante fue la fundación del Consorcio de Linux Embebido (ELC, del inglés Embedded Linux Consortium) conformada por corporaciones como Intel e IBM cuyo objetivo fue facilitar el uso de Linux y de código abierto en áreas de sistemas embebidos. Ese mismo año el Laboratorio de Desarrollo de Código Abierto (OSDL, del inglés Open Source Development Lab) es fundado por HP, Intel, IBM y NEC con el objetivo de dar soporte empresarial a soluciones Linux.

Año 2001

El gran anuncio de este año fue la liberación del kernel 2.4, el cual luego fue adoptado en la mayoría de las distribuciones embebidas de Linux.

- Sharp Electronics introduce PDAs basadas en Linux.
- Se libera uClibc 0.9.8, uClibc es una parte integral de la mayoría de las distribuciones de Linux embebido.
- Fue conformado el Consorcio Eclipse, formado por grandes compañías como IBM, SuSE, Red Hat, Borland con el objetivo de proveer un entorno de desarrollo para sistemas embebidos.

Año 2002-2004

Se produce el mayor avance de Linux en el mercado de sistemas embebidos.

- Motorola anuncia su teléfono móvil A760 que utiliza Linux como sistema operativo embebido.
- Linux alcanza mayor penetración en el mercado de dispositivos de redes como gateways, routers, y equipos inalámbricos SOHO (del inglés, Small Office-Home Office).

El avance de GNU/Linux como sistema operativo embebido en los últimos años ha sido adoptado por la mayoría de las compañías. AMD, ARM, TI, Motorola, Intel, entre otras, han preferido utilizar GNU/Linux en sus diferentes plataformas de hardware. Esta tendencia se ve reflejada en el estudio realizado en los últimos años por Linuxdevices.

La breve reseña sobre la evolución de GNU/Linux como sistema operativo embebido da cuenta de la tendencia del mercado de los sistemas embebidos.

1.9 Compilador

Un compilador acepta programas escritos en un lenguaje de alto nivel y los traduce a otro lenguaje, generando un programa equivalente e independiente, que puede ejecutarse tantas veces como se quiera. Este proceso de traducción se conoce como compilación. En un compilador hay que distinguir tres lenguajes diferentes:

· El de los programas de partida (LA)

· El de los programas equivalentes traducidos (LB), normalmente el lenguaje de máquina.

· El lenguaje en que está escrito el propio compilador (LC), que puede ser igual o diferente a uno de los otros dos.

Los programas interpretados suelen ser más lentos que los compilados, pero los intérpretes son más flexibles como entornos de programación y depuración. Comparando su actuación con la de un ser humano, un compilador equivale a un traductor profesional que, a partir de un texto, prepara otro independiente traducido a otra lengua, mientras que un intérprete corresponde al intérprete humano, que traduce a viva voz las palabras que oye, sin dejar constancia por escrito.

Partes de un compilador:

Normalmente los compiladores están divididos en dos partes:

Front End*:* es la parte que analiza el código fuente, comprueba su validez, genera el árbol de derivación y rellena los valores de la tabla de símbolos. Esta parte suele ser independiente de la plataforma o sistema para el cual se vaya a compilar.

Back End*:* es la parte que genera el código máquina, específico de una plataforma, a partir de los resultados de la fase de análisis, realizada por el Front End.

Esta división permite que el mismo Back End se utilice para generar el código máquina de varios lenguajes de programación distintos y que el mismo Front End que sirve para analizar el código fuente de un lenguaje de programación concreto sirva para generar código máquina en varias plataformas distintas.

El código que genera el Back End normalmente no se puede ejecutar directamente, sino que necesita ser enlazado por un programa enlazador (linker).

Tipos de compiladores:

Compiladores cruzados: generan código para un sistema distinto del que están funcionando.

Compiladores optimizadores: realizan cambios en el código para mejorar su eficiencia, pero manteniendo la funcionalidad del programa original.

Compiladores de una sola pasada: generan el código máquina a partir de una única lectura del código fuente.

Compiladores de varias pasadas: necesitan leer el código fuente varias veces antes de poder producir el código máquina.

Compiladores JIT (Just In Time): forman parte de un intérprete y compilan partes del código según se necesitan.

1.9.1 Compilador Cruzado

Un **compilador cruzado** es un compilador capaz de crear código ejecutable en otra plataforma distinta a aquella en la que él se ejecuta. Esta herramienta es útil cuando quiere compilarse código para una plataforma a la que no se tiene acceso, o cuando es incómodo o imposible compilar en dicha plataforma (como en el caso de los sistemas empotrados).

Un ejemplo de un compilador con estas posibilidades es el NASM, que puede ensamblar, entre otros formatos, ELF (para sistemas UNIX) y COM (para DOS).

1.9.2 Compilador para GNU/Linux a utilizar

¿Qué es GCC?

Las siglas GCC significan *GNU Compiler Collection* (Colección de compiladores GNU). Antes estas siglas de *GNU C Compiler* (Compilador C GNU). Como su nombre indica es una colección de compiladores y admite diversos lenguajes: C, C++, Objetive C, Chill, Fortran, y Java.

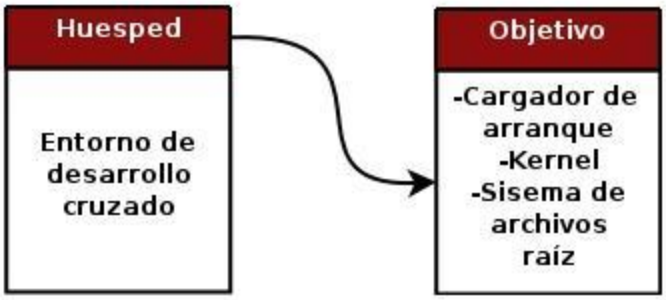
El compilador se distribuye bajo la licencia GPL (General Public License) lo que lo hace de libre distribución: se pueden hacer copias de él y regalarlas o venderlas siempre que se incluya el código fuente (o se indique cómo conseguirlo) y se mantenga la licencia.

Existen versiones para prácticamente todos los Sistemas Operativos. Viene incluido en la mayoría (si no en todas) las distribuciones de GNU/Linux. En el desarrollo de este compilador participan cientos de voluntarios de todo el mundo.

1.9.3 Compilación Cruzada

La compilación de código fuente que realizada bajo una determinada arquitectura genera código ejecutable para una arquitectura diferente se denomina **compilación cruzada**. Para realizar este tipo de compilación es necesario contar con una serie de programas y librerías que establezcan un ambiente propicio para llevar a cabo esta tarea. Este ambiente se denomina **entorno de compilación cruzada**.

Existen dos tipos de sistemas, el **Huésped**, es en donde se realiza la compilación del sistema, y el **Objetivo** donde se ejecuta el código.



1.9.4 Entorno de compilación cruzada

Para implementar un entorno de compilación cruzada es necesario un conjunto de librerías, utilitarios y binarios. En la bibliografía relacionada con sistemas GNU/Linux embebidos a este conjunto de componentes se les denomina ***toolchain components***.

Estos componentes son:

- Compilador C: compilador de C básico, generador de código objeto (tanto del kernel como de aplicaciones).
- Librería C: implementa las llamadas al sistema mediante APIs.
- Binutils: conjunto de programas necesarios para la compilación, enlazado, ensamblado y depuración de código. Entre otros, los binarios principales son: ld (GNU linker), as (GNU assembler).

1.10 Conclusiones

En este capítulo quedaron expuestos los principales elementos para la realización de un sistema embebido. Por otra parte se realizó un estudio de las SBC existentes para el encaminamiento de paquetes, se abordaron los pasos a utilizar para la creación de un sistema embebido, así como los compiladores y las librerías.

Capítulo

2

2-Procedimiento para realizar el entorno de compilación cruzada.

2.1 Introducción

En este capítulo se explicará más detalladamente cómo es el proceso de compilación cruzada, la conexión que se establece entre el sistema huésped y el objetivo, se hablará de la herramienta Buildroot, utilizada para realizar el entorno de compilación cruzada.

2.2 Sistema Huésped

Como hemos descrito en el capítulo 1, la implementación de un entorno de compilación cruzada nos brinda la posibilidad de aprovechar los recursos que disponemos en una computadora tipo PC.

Esta tarea se ha llevado a cabo sobre una PC de escritorio ASUS P5LD2-VM con las siguientes características:

- **CPU :** LGA775 socket for Intel® Core™2 Extreme / Core™2 Duo / Pentium® D /, Pentium® 4 / Celeron® D Processors, Intel® EM64T / EIST / Hyper-Threading Technology, **Only PCB R2.0(or higher) support Intel® Core™2 processor.*
- **Chipset:** Intel 945G, Intel ICH7.
- **Front Side Bus:** 1066/800/533 MHz.
- **Memory**: 4 x DIMM, max 4GB, DDR2 667 / 533, non-ECC, un-buffered memory Dual Channel Architecture. Due to general PC architecture, a small amount of memory is reserved for system usage and thus the actual memory size is less than the stated amount.
- **Expansion Slots:** 1 x PCI-E x16, 1 x PCI-E x1, 2 x PCI, PCI 2.2.
- **VGA:** Intel Graphics Media Accelerator 950.
- **Storage/RAID:** Intel ICH7 South Bridge: 1 x UltraDMA 100/66/33, 4 x Serial ATA 300/150, ITE IDE controler: 1x UltraDMA 133/100.
- **Audio:** Realtek ALC 882, 8-ch High-Definition Audio CODEC S/PDIF out interface.
- **USB:** 8 USB2.0 ports.
- **Back Panel I/O Ports:** 1 x Parallel, 1 x Serial, 1 x PS/2 Keyboard, 1 x PS/2 Mouse, 1 x VGA, 1 x Audio I/O, 1 x RJ45, 4 x USB.
- **Internal I/O Connectors:** 2 x USB connectors supports additional 4 USB ports, CPU / Chassis FAN connectors, 24-pin EATX Power connector, 4-pin ATX 12V Power connector, Chassis Intrusion connector, S/PDIF out connector, CD audio in, 3-pin PWR FAN connector, Front panel High Definition Audio connector.**(7)**

2.3 Sistema Objetivo

La arquitectura del sistema objetivo para la cual se desarrolló la compilación cruzada, es la de una Routerboard 230 con las siguientes características:

- **CPU:** 266 Mhz NSC SC1100 system on a chip CPU (Pentium MMX architecture)
- **Memory Slot:** SoDIMM (up to 512MBytes SDRAM)
- **Harddrive connectors:** CompactFLASH I/II socket (support for standard CF and IBM Microdrive)
  44 pin boxhead IDE connector for Laptop Hard Drive (2.5 inch).
- **Ethernet ports:** Two 10/100 Mb/s Ethernet using the NSC DP83816 (DP83815 driver compatible) one of them with Power over Ethernet 802.3af standard.
- **Serial ports:** One port with DB9 standard.
- **USB port:** One port with USB 1.1 standard.
- **Mini PCI slot:** One slot with Type III standard.
- **PCI slot:** One slot with universal support (+/-12v, 5v, 3.3v).
- **PCMCIA slot:** Dual PCMCIA/CardBUS.
- **BIOS:** 2 Mbit Flash BIOS on board.
- **Power connections:** Onboard power jack 20-56vDC in, Onboard power header 48v in (to connect elecom 48v power wires), 3.3v out power header, 5v out power header. **(4)**

2.4 Comunicación entre sistema Huésped y Objetivo

La transferencia del sistema GNU/Linux desde la PC hacia el Sistema Objetivo se puede realizar mediante tres opciones:

1- Un medio de almacenamiento externo:

Más precisamente almacenaremos el sistema en una memoria compact flash de 128 Mb de capacidad, conectada a la computadora mediante un grabador de compact flash a través de un puerto USB. Una vez realizada la transferencia completa del sistema, se conectará al slot compact flash del Sistema Objetivo, para que el mismo realice el arranque del sistema desde este medio. En la figura 1 observamos una tarjeta compact flash.

Fig.1.Compact Flash de 128 Mb

2 - Arranque a través de la red:

Es decir, el equipo arranca buscando por FTP una imagen del sistema para iniciar. Esta alternativa es muy útil en las primeras etapas, ya que se evita la conexión y desconexión de la tarjeta de memoria, evitando desgastes y roturas en los conectores.
Una vez que el sistema se encuentra estable, es preferible utilizar la tarjeta de memoria y realizar las configuraciones extras utilizando una consola terminal.
Para configurar la Routerboard 230 y seleccionar el modo de arranque, se ha utilizado un cable denominado módem-nulo que se conecta desde el puerto serie de la PC a la interfaz RS-232 del dispositivo.

3-Arranque del disco duro portátil:

El sistema objetivo posee un conector para un disco duro portátil de 2.5 pulgadas, al cual se le graba el sistema para que inicie el booteo desde este. La información se graba a través de un cable conversor de puerto Mini IDE a IDE.

2.5 Componentes toolchains

Los componentes necesarios para implementar el entorno de compilación cruzada consisten básicamente en tres elementos:

- Compilador C : compilador de C básico
- Librería C: implementa las llamadas al sistema mediante APIs.
- Binutils: conjunto de programas para compilación, enlazado, ensamblado y depuración de código.

Cada una de estas herramientas funciona en dependencia de las otras, es por esto que configurar y compilarlas es una operación delicada. La complejidad radica en que no existe una compatibilidad asegurada entre las diferentes versiones de los componentes. Por esto, la primera etapa consiste en seleccionar las versiones de cada uno de estos componentes compatibles entre sí.

La combinación de diferentes versiones ha dado resultados no esperados. Suele ser frustrante la repetición de estos pasos una y otra vez, y que luego de horas de compilación aparezcan errores de todo tipo.

Las versiones de las herramientas con las que se ha obtenido una compilación exitosa son las siguientes:

- Compilador: gcc 4.3.2
- Librería C: uClibc 0.9.30
- Binutils 2.19

Este conjunto de herramientas se han configurado y compilado utilizando un paquete de software denominado **Buildroot**.

Buildroot nos brinda una interfaz amigable para seleccionar las versiones de las toolchains y configuraciones extras para implementar el entorno de compilación cruzada. Este software nos permite seleccionar a través de diferentes menús las características que son necesarias de implementar.

2.6 Buildroot

Específicamente, Buildroot es un conjunto de makefiles y parches que permiten generar tanto el entorno de compilación cruzada como el sistema de archivo raíz para el sistema objetivo, otorgando una manera organizada de realizar las configuraciones. Por las características mencionadas es una alternativa muy utilizada para el trabajo con sistemas GNU/Linux embebido.

2.6.1 Estructura interna

Buildroot es básicamente un conjunto de Makefiles que descargan, configuran y compilan un determinado software. Esto incluye aplicar parches para diferentes paquetes de software, principalmente los involucrados en las herramientas para la compilación cruzada (toolchain). Básicamente hay un archivo Makefile por software, denominados con la extensión **.mk**. Los Makefiles están divididos y ubicados en 4 secciones (directorios):

- **project**: contiene Makefiles y archivos asociados para todo el software relacionado con la compilación del sistema de archivo raíz.
- **toolchain**: contiene Makefiles y archivos asociados para todo el software relacionado con las herramientas de compilación cruzada: binutils, ccache, gcc, gdb, kernel-headers, y uClibc.
- **package**: contiene Makefiles y archivos asociados para todas aquellas herramientas de espacio de usuario que Buildroot puede compilar y agregar al sistema de archivo raíz. Existe un subdirectorio por herramienta.
- **target**: contiene Makefiles y archivos asociados para el software relacionado con la generación de la imagen del sistema de archivos raíz. Cuatro tipo de sistemas de archivos son soportados: ext2, jffs2, cramfs y squashfs. Para cada uno de estos existe un subdirectorio con los archivos necesarios. También hay un directorio default que contiene la estructura del sistema de archivos objetivo.

2.6.2 Funcionamiento

Cada directorio contiene al menos dos archivos:

- **.mk** que es el Makefile que descarga, configura, compila e instala el software.
- **Config.in** archivo de descripción de la herramienta de configuración. Describe las opciones relacionadas con el software seleccionado o en uso.

El Makefile principal realiza los siguientes pasos:

1- Crea el directorio de descarga (dl). Aquí es donde los tarballs serán descargados. Es interesante conocer que se encuentran en este directorio debido a que puede ser útil guardarlos para evitar futuras descargas.

2- Crea el directorio compartido (build_ARCH, donde ARCH es nuestra arquitectura). Es aquí donde todas las herramientas no configurables del espacio de usuarios serán compiladas.

3- Crea el directorio específico del proyecto de compilación (project_build_ARCH/$PROJECT). Es aquí donde todas las herramientas configurables del espacio de usuario son compiladas.

4- Crea el directorio donde los resultados específicos del proyecto se almacenan (binaries/$PROJECT). Es aquí donde la imagen del RFS es guardada, además es utilizada para almacenar la imagen del kernel Linux y cualquier utilitario, bootloaders, etc. necesarios para el sistema objetivo.

5- Crea el directorio de compilación de las toolchains (toolchain_build_ARCH). Acá es donde se compila las toolchains de la compilación cruzada.

6- Instala el directorio de la plataforma (build_ARCH/staging_dir). Aquí es donde las toolchains de la compilación cruzada son instaladas.

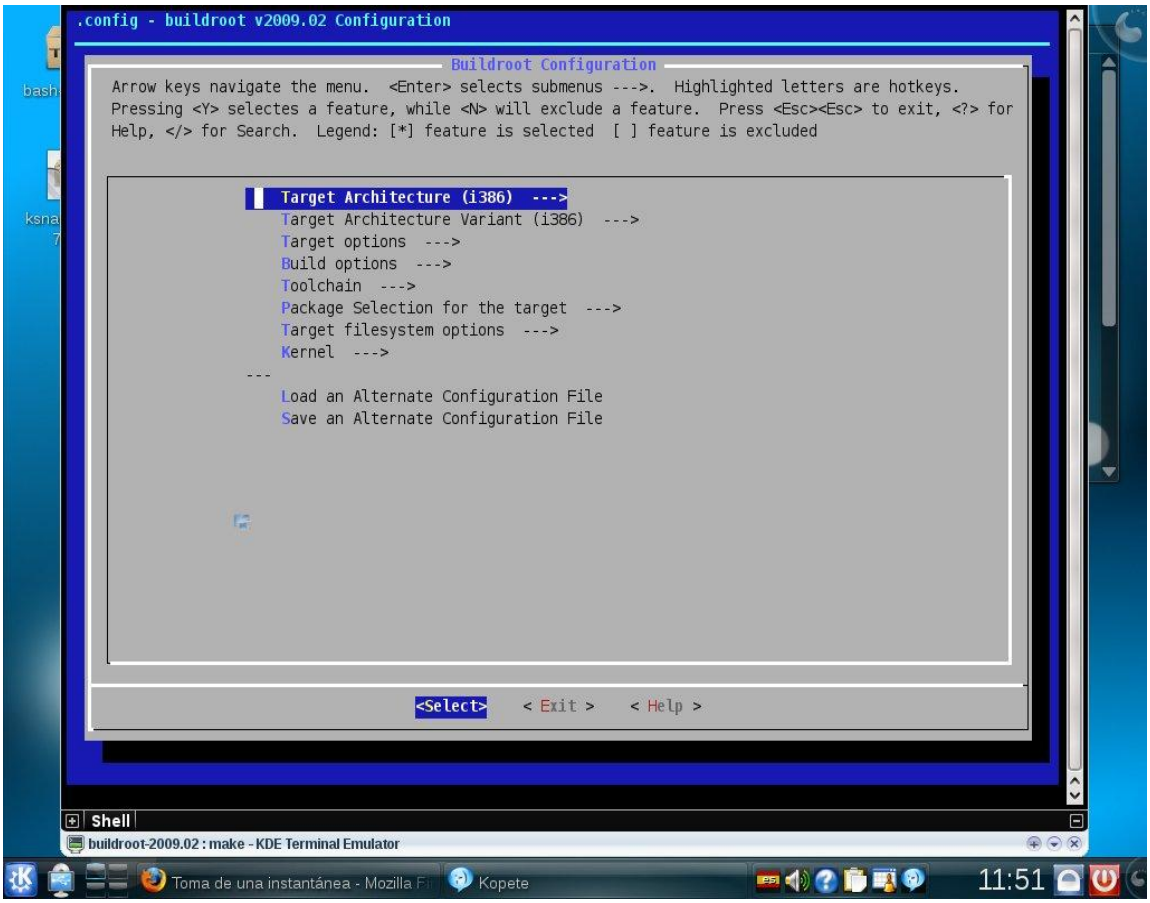
7- Crear el directorio target y la estructura del sistema target (project_build_ARCH/root). Aqui se encuentra el directorio que contiene el sistema de archivos final.

8- Resuelve dependencias para los paquetes seleccionados.

En base a la arquitectura de hardware que hayamos seleccionado se remplaza ARCH por la arquitectura seleccionada**.**

2.6.3 Configuración y uso

El menú de opciones y el modo de realizar la compilación es idéntica a la compilación y configuración del kernel Linux, incluso los comandos para iniciar el proceso son los mismos. Para comenzar el proceso de configuración es necesario ubicarse dentro del directorio donde se ha descomprimido la descarga de Buildroot y luego se debe ejecutar el siguiente comando, embedded@linux:~/home/linux/buildroot $ **make menuconfig**



En la Fig.2 se observa el menú principal de configuración de Buildroot.

En los siguientes puntos se describen las opciones de configuración que provee la interfaz de Buildroot y se detallan las alternativas seleccionadas.

Target architecture

Aquí se define la arquitectura que posee el microprocesador del sistema objetivo. Para el caso del Routerboard 230 la opción a seleccionar es **i386** (Fig.3). De las variantes de esta arquitectura se eligió la Pentium MMX (Fig.4).

Target options

Aquí se define algunas opciones como el nombre del proyecto, nombre del sistema y mensaje de bienvenida del sistema (Fig.5).

Build options

En esta sección se encuentra la configuración general (Fig.6). Se pueden modificar el nombre de los directorios de descarga, el método y comando para realizar la descarga desde internet, los mirrors de Sourceforge, los nombres de los comandos y otras opciones globales.

Toolchain

Además de utilizar Buildroot como generador de las herramientas de compilación cruzada, en este punto es posible utilizar herramientas externas. Bajo esta opción se puede seleccionar la configuración y versiones del conjunto de toolchains (binutils, gcc, librería C), los headers del kernel, herramientas de depuración tanto para el sistema objetivo como para el sistema huésped (Fig.7). En base al Routerboard 230 se han seleccionado las siguientes opciones.

- gcc 4.3.2
- uClibc 0.9.30
- binutils 2.19
- kernel headers 2.6.28

Package selection for the target

Aquí se pueden seleccionar los binarios deseados para el sistema raíz objetivo, herramientas de desarrollo y utilizar una versión reducida de **Busybox** para generar el sistema de archivos raíz con sus binarios por defecto. En este punto se ha seleccionado esa opción para poder aprovechar las funciones de esta herramienta, la cual explicaremos más adelante (Fig.8).

Target filesystem options

En este punto se hace la selección del tipo de sistema de archivos para el sistema objetivo (Fig.9). Aquí se ha seleccionado ext2 como sistema de archivos.

Kernel

Aquí se puede seleccionar la versión del kernel a utilizar, se pueden hacer algunas configuraciones del mismo y se puede establecer el formato del binario de este kernel, en nuestro caso seleccionamos bzImage (Fig.10). **(2)**

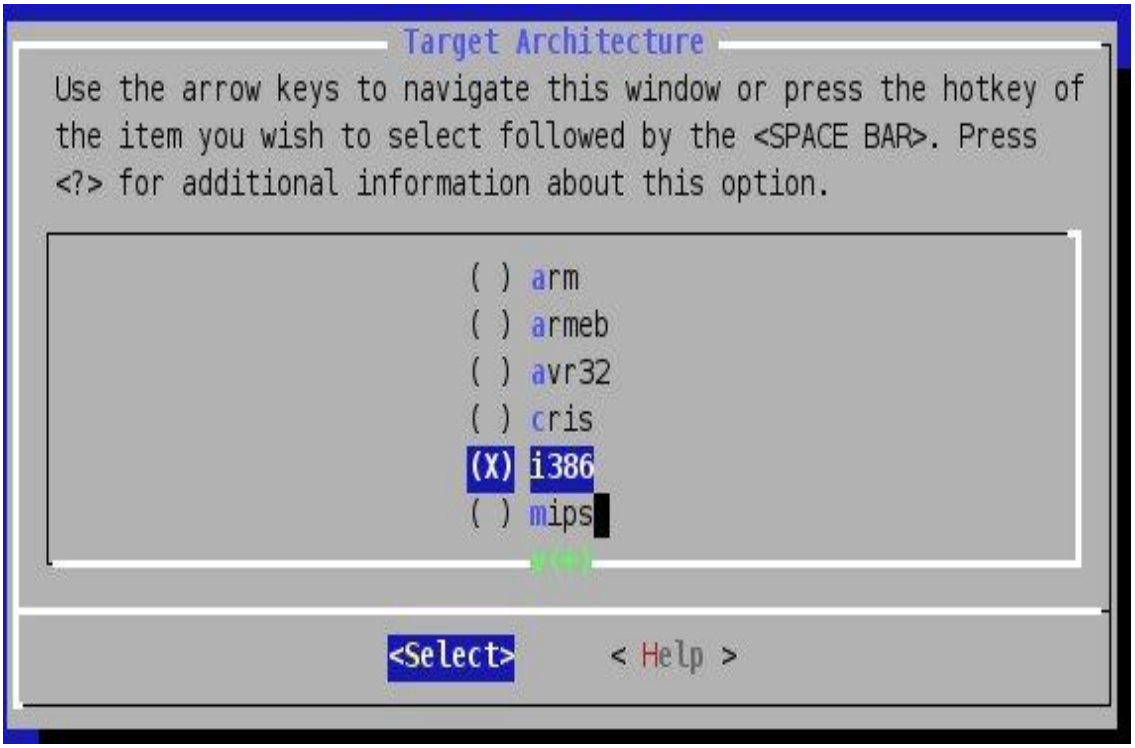


Fig.3: Arquitectura i386

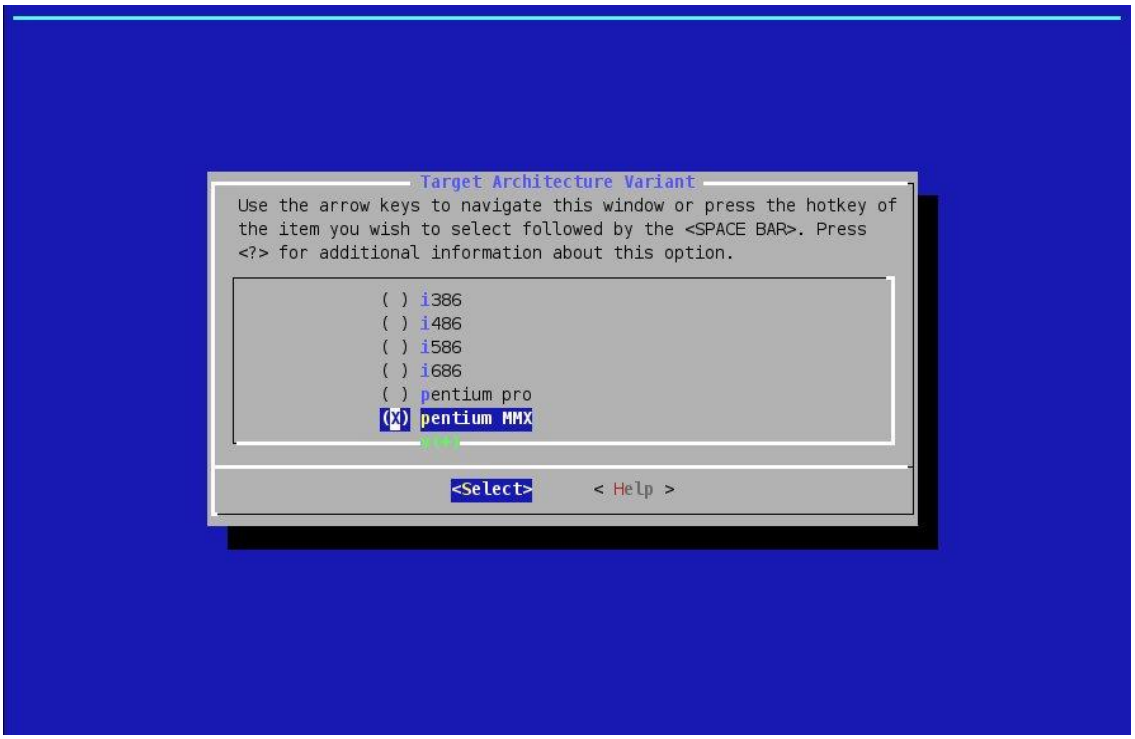


Fig.4: Variante de Arquitectura

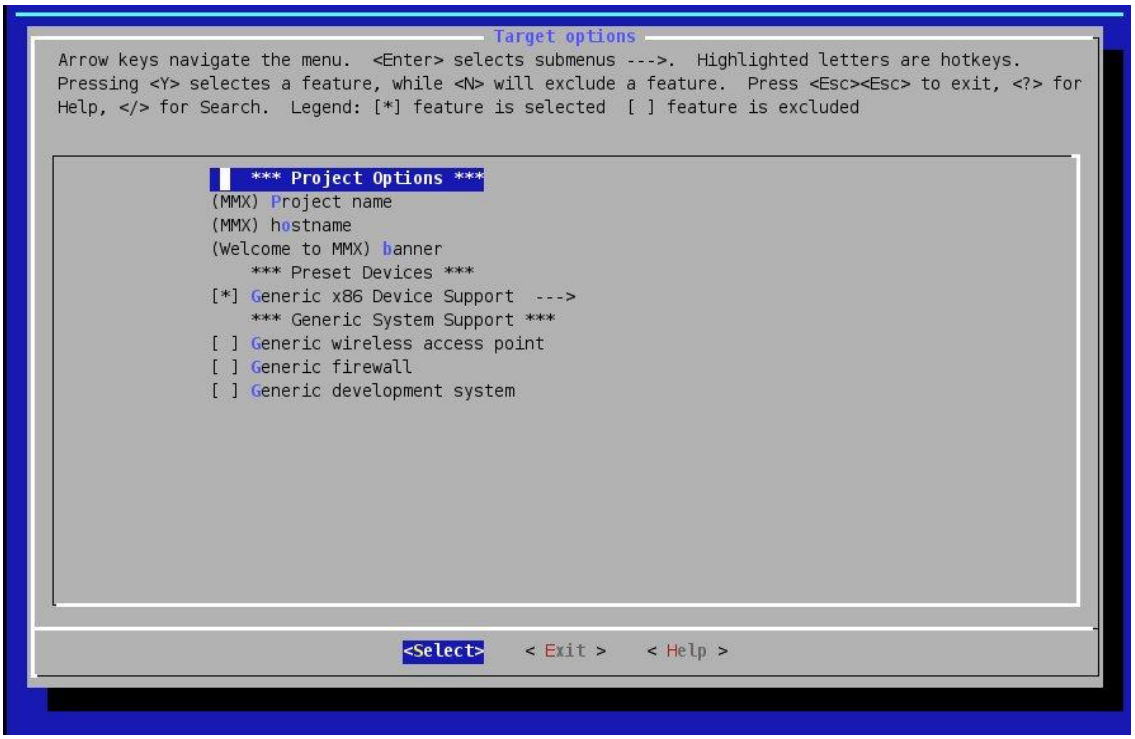


Fig.5: Target options

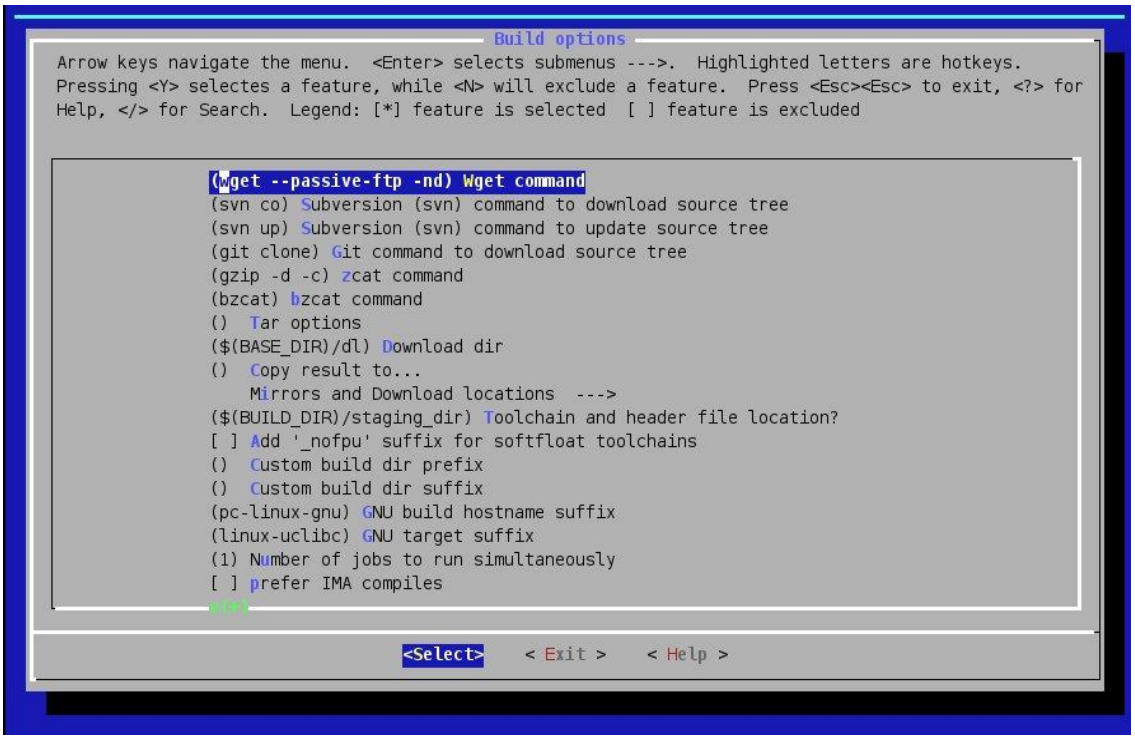


Fig.6: Build options

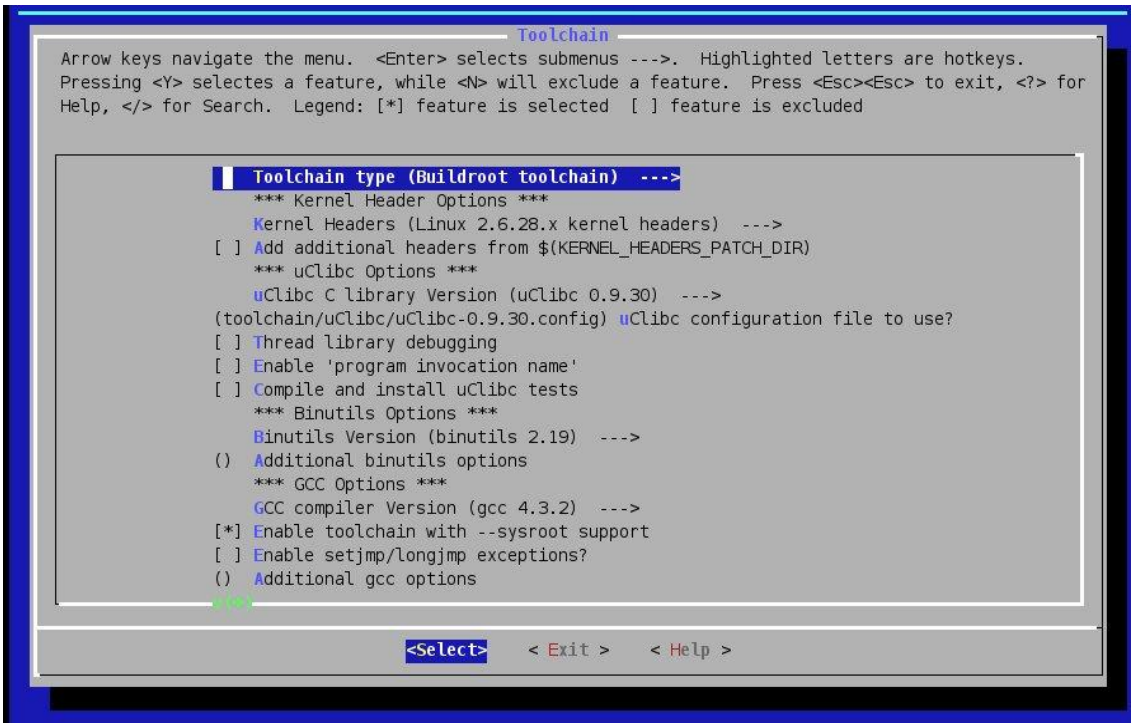


Fig.7: Toolchain

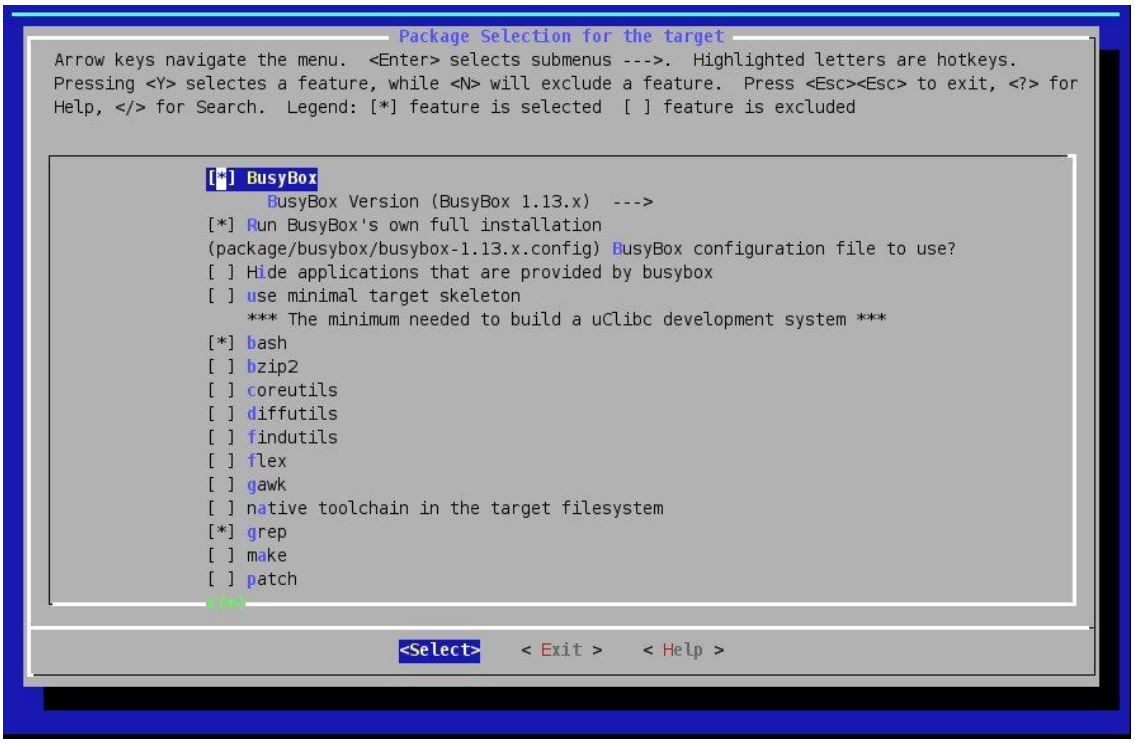


Fig.8**:** Package selection for the target

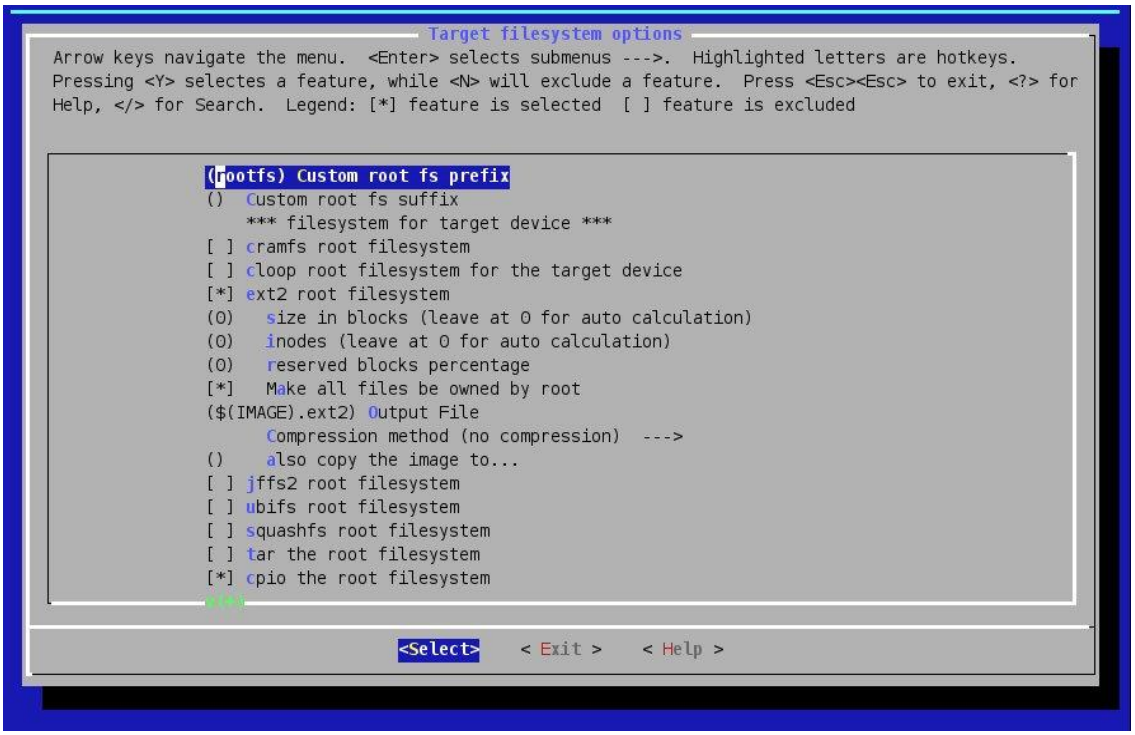


Fig.9: Target filesystem options

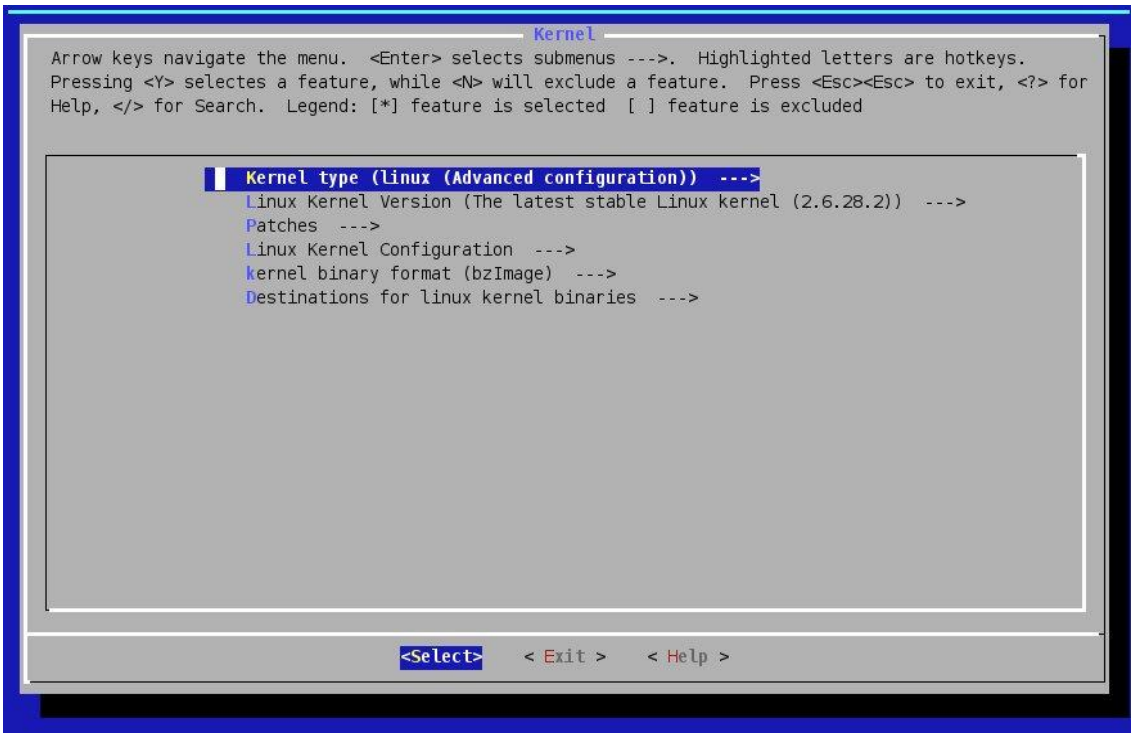


Fig.10 Kernel

Luego del proceso de configuración que se ha descrito en los puntos anteriores y una vez guardados los cambios se procede con la etapa de compilación. Simplemente se realiza el comando ***make*** estando ubicado en el directorio principal del Buildroot. Esto inicia el proceso de compilación de las herramientas de compilación cruzada. Dependiendo de la velocidad de procesamiento del equipo huésped y de la velocidad de la conexión a internet el proceso de compilación puede tardar algunas horas. El proceso muestra en su etapa inicial los siguientes mensajes:

embedded@linux:~/home/linux/buildroot $ **make**

Checking build system dependencies:
CC clean: Ok
CXX clean: Ok
CPP clean: Ok
CFLAGS clean: Ok
INCLUDES clean: Ok
wget --passive-ftp -P /home/linux/buildroot /dl

```
http://www.uclibc.org/downloads/toolchain/linux-libc-headers-2.4.31.tar.bz2
Petición HTTP enviada, esperando respuesta... 200 OK
100%[===================================================================>]
4.194.659     93.31K/s    ETA 00:00
```

Si bien se ha destacado que Buildroot permite generar el sistema de archivos raíz a ser utilizado en el sistema objetivo, aquí se ha realizado con otra herramienta específica para esta tarea denominada BusyBox.

La función fundamental del Buildroot es permitirnos obtener un entorno de compilación cruzada funcional para realizar la compilación del sistema y de programas a ser ejecutados en el dispositivo Routerboard 230, brindando una interfaz amigable o al menos más intuitiva que la realización de una configuración manual de los parámetros en cada archivo de configuración. Si bien Buildroot brinda un ambiente organizado y simplificado a la hora de realizar un entorno de compilación cruzada, muchas veces ha sido necesario modificar desde los archivos de configuración fuente debido a errores en el proceso de compilación, por eso es útil conocer cuáles son estos archivos.

Los archivos de configuración relevantes son los siguientes:

- **buildroot/.config** Bajo este archivo se encuentra la configuración global del Buildroot, modificar este archivo es equivalente a seleccionar las opciones que provee la interfaz de comando *make menuconfig*.
- **buildroot/toolchain/uClibc/uClibc-0.9.30.config** Aquí se encuentran parámetros relacionados con la librería uClibc, con el tipo de microprocesador del dispositivo objetivo, como el endianismo, la cantidad de bits, si posee unidad de punto flotante, entre otros.
- **buildroot/toolchain_build_arch/uClib-0.9.30/.config**

Bajo el directorio buildroot/package/ se encuentran los subdirectorios con el nombre de cada paquete, dentro de estos se pueden encontrar los archivos con extensión .mk que poseen la información sobre las dirección web de donde será descargado, el número de versión y otra información modificable.

2.7 Busybox

Es un programa que combina muchas utilidades estándares de Unix en un solo ejecutable pequeño. Es capaz de proveer la mayoría de las utilidades que están especificadas para los sistemas Unix además de muchas de las utilidades que esperas ver en los sistemas GNU/Linux. Busybox es utilizada normalmente en sistemas que funcionen desde un disco flexible o en sistemas Linux Empotrado. BusyBox se ha escrito con la optimización de tamaño y recursos limitados en mente. También es muy modular, de modo que puede incluir o excluir los comandos (o elementos) en tiempo de compilación. Esto hace que sea fácil de personalizar sus sistemas embebidos. Es software libre licenciado bajo la licencia GNU GPL, fue definido como "*la navaja suiza de los sistemas Linux empotrados*".

El programa fue escrito originalmente por Bruce Perens en 1996, con el objetivo principal de poner un sistema completo en un disco flexible que pueda iniciar la máquina y funcionar como disco de rescate e instalador para la distribución Debian GNU/Linux.

Los comandos de Busybox proporcionan únicamente las funcionalidades más usadas, estas se pueden invocar mediante la emisión de un comando como argumento en la línea de comandos. Por ejemplo: **/bin/busybox ls**

Por supuesto, añadiendo "/ bin / busybox" en cada comando sería doloroso. Por lo tanto, la mayoría de las personas invocan BusyBox utilizando enlaces a los binarios de BusyBox. Por ejemplo: **ln-s / bin / busybox ls**

. / ls

2.8 Uso del Entorno de Compilación

Si se desea, se puede compilar nuestros propios programas u otro software que no estén envasados en Buildroot. Para ello, se puede utilizar las herramientas que se generó por Buildroot.
La herramienta elaborada por Buildroot por defecto se encuentra en build_ARCH / staging_dir /. La forma más sencilla de usarlo es agregar build_ARCH / staging_dir / usr / bin / a la variable

de entorno PATH y, a continuación, utilizar el arco-linux-gcc, arco-linux-objdump, arco-ld-linux, entre otros.Por ejemplo:

export PATH = "$ PATH: ~ / buildroot / build_arch / staging_dir / usr / bin /"
A continuación, puede hacer:

arch-linux-gcc-o foo foo.c

2.9 Conclusiones

En este capítulo se definieron las propuestas reales del sistema huésped y del objetivo, además se explicó la herramienta utilizada que es el Buildroot, así como su estructura interna y funcionamiento. Se tocaron temas como el uso que puede tener el entorno de compilación cruzada y la herramienta de Busybox, la cual es muy importante ya que nos aporta los comandos necesarios para interactuar con el sistema.

Capítulo

3

3-Creación de la distribución personalizada.

3.1 Introducción

En este capítulo se explicará como es el proceso de creación de la distribución, se abordarán temas como son: herramientas utilizadas para la elaboración del sistema, gestor de arranque utilizado en la distribución y los pasos para realizar el empotramiento de la distribución en la Routerboard 230.

3.2 Herramientas utilizadas para la creación del sistema.

1- **Bash:**

Es un shell de Unix (**intérprete de órdenes de Unix**) escrito para el proyecto GNU. Su nombre es un acrónimo de ***b****ourne-****a****gain* ***sh****ell* (**otro shell bourne**) — haciendo un juego de palabras (born-again significa renacimiento). Bash es el intérprete predeterminado en la mayoría de sistemas GNU/Linux, además de Mac OS X Tiger, y puede ejecutarse en la mayoría de los Sistemas Operativos tipo Unix. También se ha llevado a Microsoft Windows por el proyecto Cygwin.

2- **Iptables:**

Es la línea de comandos de usuario que utiliza para configurar el Linux 2.4.x y 2.6.x las reglas de filtrado de paquetes IPv4. Está dirigido a los administradores de red. Contiene traducción de direcciones de red, ya que también está configurado desde el filtro de paquetes de reglas. El paquete también incluye iptables e ip6tables, este último se utiliza para configurar el filtrado de paquetes IPv6.

Dependencias

Iptables requiere un núcleo que cuenta con el filtro de paquetes ip_tables. Esto incluye todos los 2.6.x del kernel 2.4.x y las emisiones.

Características Principales

Lista, añade, elimina y modifica el contenido de reglas en el filtro de paquetes de reglas

3- Iproute2:

Es un paquete de utilidades desarrollado por Alexey Kuznetsov. Este paquete es un conjunto de herramientas muy potentes para administrar interfaces de red y conexiones en sistemas Linux. Este paquete reemplaza completamente las funcionalidades presentes en ifconfig, route, y arp y las extiende llegando a tener características similares a las provistas por dispositivos exclusivamente dedicados al ruteo y control de tráfico. Este paquete lo podemos encontrar incluido en distribuciones de Debian y RedHat con versiones del núcleo mayores a 2.2.

Funcionalidades

Algunas de las funcionalidades principales que provee iproute2 son:

- **QoS (Quality of service)**, priorizando distintos tipos de tráfico.
- **Múltiples tablas de ruteo** por diferentes puertas de enlaces conectadas a distintos dispositivos.
- **Balanceo de carga**, asignándole pesos a cada una de las placas existentes dentro de mi máquina.
- **Definición de túneles**, los objetos tunnel son túneles que encapsulan paquetes en un formato IPv4 y se envían por la infraestructura IP.

4- Udev:

Es el gestor de dispositivos que usa el kernel Linux en su versión 2.6. Su función es controlar los ficheros de dispositivo en /dev. Es el sucesor de devfs.

Motivación

En un sistema Linux tradicional (sin udev ni devfs), en el directorio /dev hay nodos de dispositivo creados para cada dispositivo conocido, esté o no en el sistema. Se dice que es un conjunto de ficheros estático, ya que los nodos no cambian.

Además, la forma de acceder a un periférico concreto no es siempre la misma, ya que depende de qué otros aparatos hay conectados: si se conectan los discos A y B, se llamarán disco1 y disco2 respectivamente. Pero si está sólo un disco (el B, por ejemplo), se llamará disco1, porque sólo hay uno. El B ha cambiado de nombre.

Este modelo de gestión de dispositivos da algunos problemas:

El directorio /dev es enorme y difícil de manejar, ya que incluye todos los dispositivos posibles

Los números mayor y menor que se asocian a cada dispositivo se estaban acabando

Los usuarios necesitan que cada dispositivo sea accesible de la misma manera; no aceptarán que por conectar un disco USB al sistema tengan que reconfigurar la cámara de vídeo.

Los programas necesitan poder detectar cuándo se ha conectado o desconectado un dispositivo, y cuál es la entrada que se le ha asociado en /dev.

Udev soluciona estos problemas, sobre todo el de poder acceder a un dispositivo con un nombre siempre fijo. Ésta fue la razón por la que se hizo udev, ya que antes estaba devfs, que solucionaba alguno de estos problemas, pero no todos.

Características

Sólo dispositivos conectados: udev mantiene en /dev sólo las entradas correspondientes a los dispositivos que hay conectados al sistema. Así se soluciona el problema del /dev superpoblado.

No se usa mayor y menor: No se usa el número mayor y menor para reconocer a cada dispositivo. Puede funcionar incluso aunque estén elegidos al azar. Por tanto, no le afecta el que se acaben las combinaciones mayor/menor asignables.

Permite dar nombre fijo: Permite dar un nombre fijo para cada dispositivo, por ejemplo cámara, sin que éste dependa de qué otros dispositivos hay conectados ni del orden en que se

han conectado. Un disco duro, por ejemplo, se reconoce por el identificador de su sistema de ficheros, el nombre del disco, y el conector físico en el que está.

Avisa a los programas: Avisa mediante mensajes D-BUS para que cualquier programa del espacio de usuario pueda enterarse cuando un dispositivo se conecta o desconecta. También permite a los programas consultar la lista de dispositivos conectados y la forma de acceder a cada uno.

Todo en espacio de usuario: udev hace que toda la política de nombres de dispositivo esté en espacio de usuario, y no en el kernel. Esto hace posible que un programa cualquiera pueda decidir el nombre de un dispositivo, por ejemplo basándose en sus características. No es necesario modificar el kernel si no se está conforme el nombre que se le da a un dispositivo. Otra ventaja de tener el demonio de udev (udevd) en espacio de usuario es que esta memoria se puede llevar a disco (al espacio de intercambio), a diferencia de la memoria de kernel. Esto lo hace apropiado para sistemas embebidos con poca memoria física.

Otras características

Udev respeta la forma de nombrar dispositivos definidos en el Linux Standard Base (LSB), aunque permite que los usuarios usen otros nombres.

El proceso (udevd) ocupa poca memoria y no necesita ejecutarse siempre. Esto también favorece a los sistemas embebidos y equipos poco potentes.

Diferencias con devfs

Devfs también se hizo para solucionar algunos de los problemas originales, pero no todos. Los autores de udev explican las razones por las que **udev** es mejor que **devfs**.

Resumiendo:

- devfs sólo muestra en /dev los dispositivos conectados, al igual que udev
- devfs no soluciona el problema de los números mayor/menor que se acaban, ya que no permite que sean dinámicos
- devfs no permite dar un nombre fijo a cada dispositivo (ésta es la razón por la que se hizo udev)

- devfs, al igual que udev, permite que los programas sepan cuándo se conecta o desconecta un dispositivo, mediante consultas a un proceso demonio
- en devfs la política de nombres de dispositivo sigue dentro del kernel, no en los programas
- devfs no sigue el estándar de nombres definido en LSB
- devfs es pequeño, pero se encuentra en espacio de kernel, que es memoria que no se puede llevar a disco

Implementación

Udev se ejecuta mediante un demonio: el proceso udevd, que detecta cuándo se ha conectado o desconectado un dispositivo del sistema.

Cuando pasa uno de estos eventos, **udev** obtiene información de contexto (subsistema del kernel, conector físico usado, nombre dado por el kernel, etc.) y también del propio dispositivo (número de serie, fabricante, etc.).

Esta información la puede encontrar mediante los datos que hay en /sys, en donde está montado el sistema de ficheros **sysfs**, del que se encarga el kernel (versión 2.6 y posteriores).

Un conjunto de reglas (en /etc/udev/rules.d/) deciden qué acción hay que hacer a partir de los datos obtenidos. Lo que la regla hará será -probablemente- dar un nombre al dispositivo, crear el fichero de dispositivo apropiado, y ejecutar el programa que se haya configurado para que acabe de hacer funcionar el dispositivo.

Las reglas pueden asociar un nombre fijo a un dispositivo, pero también pueden llamar a un programa externo que dé más información sobre el dispositivo; así se puede conseguir un nombre más específico.

Para que puedan funcionar estos dispositivos y poder tener el control sobre ellos, es necesario usar Hotplug, que no es más que un programa que es el encargado de examinar el tipo de dispositivo y cargar los módulos que son necesarios para dejar el dispositivo listo para usar.

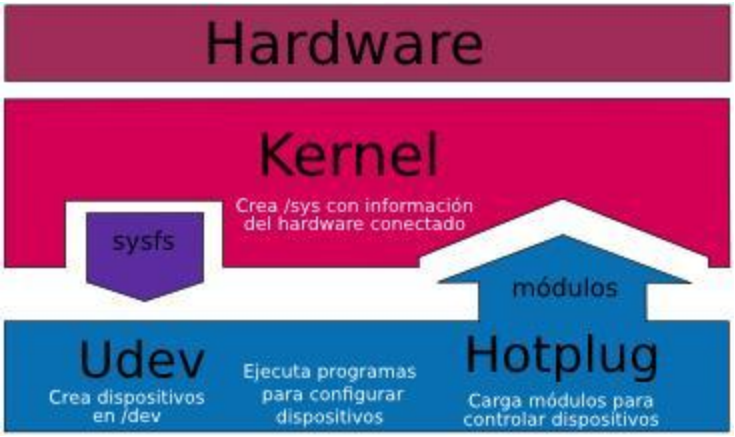


Fig.11: Proceso de control de dispositivos.

3.3 Creación de la distribución.

A continuación se detallará el proceso de creación de la distribución. Primeramente se utilizó un kernel versión 2.6.28.2 y principalmente las herramientas anteriormente explicadas, las cuales se seleccionaron en la creación del entorno de compilación. Seguidamente se ejecuta: **# make linux26-menuconfig**

Esto mostrará una interfaz que contiene todos los archivos y módulos del kernel, luego se optimiza según las características de la arquitectura del sistema objetivo (Routerboard 230).
A continuación se mencionan los drivers necesarios para el uso de la compact flash o de un disco IDE conectado a la routerboard, además de algunos drivers necesarios para el funcionamiento de la red.

Drivers de Compact Flash y discos IDE.

[*]Block devices --->

[*]Very old hard disk (MFM/RLL/IDE) driver \

<*>ATA/ATAPI/MFM/RLL support ---->

<*>gereric ATA/ATAPI disk support

[*]ATA disk support

[*] legacy /proc/ide/ support

<*>generic/default/ IDE chipset support

<*>Plataform driver for IDE interface

[*] probe IDE PCI devices in the PCI bus order

[*]Boot off-board chipsets first support

<*>Generic PCI IDE Chipset Support

<*>National Scx200 chipset support

Drivers de red

[*]Network device support --->

[*]Ethernet (10 or 100bit) ---->

--*-- Generic Media Independent Interface device support

[*]EISA, VLB, PCI and on board controllers

<*>National Semiconductor DP831x series PCI Ethernet support

<*>RealTek RLT-8139 C+ PCI Fast Ethernet Adapter support

<*>RealTek RTL-8129/8130/8139 PCI Fast Ethernet Adapter support

[*] Support for uncommon RTL-8139 rev. K

[*]Support for older RTL-8129/8130 boards

Al finalizar el proceso de optimización del kernel ejecutamos:

#make

Comenzando así el proceso de creación de la distribución, que como resultado obtendremos un binario del kernel utilizado, compactado en **bzImage** y un fichero llamado **rootfs.i586.cpio.gz** el cual hace la misma función de un initramfs, es decir es el encargado de cargar los módulos necesarios para el kernel, despojando a este de esa responsabilidad. Posteriormente se comienza el empotramiento de la distribución, se escogió la alternativa de levantar el sistema a través de una compact flash de 128 Mb, a la cual se le graban los ficheros obtenidos, esto se

hace mediante un grabador de compact flash que se puede conectar por puerto USB a una PC (Fig.12).

Fig.12 Grabador de Compact Flash

Ya ubicado el dispositivo en el conector de compact flash, comienza el **proceso de arranque** de un sistema operativo Linux, el cual se explica detalladamente a continuación. El gestor de arranque utilizado fue **EXTLINUX** porque es bastante ligero y compatible con el sistema, para instalarlo se ejecutan los siguientes comandos:

 # extlinux –i dir // dir es el directorio que usted quiera

 # touch dir/extlinux.conf

Después de creado este fichero se le debe pasar una serie parámetros como se muestra a continuación:

#nano dir/extlinux.conf

DEFAULT nombre

LABEL nombre

KERNEL /boot/bzImage

APPEND initrd=/boot/initramfs // En nuestro caso sería /boot/ rootfs.i586.cpio.gz

TIMEOUT 10

3.4 Gestores de Arranque

Un gestor de arranque es un programa que toma el control de la máquina nada más conectarse y una vez que ha terminado las verificaciones por el propio hardware de memoria y dispositivos conectados. Cuando el gestor de arranque toma el control, puede solicitar al usuario alguna información necesaria sobre qué sistema cargar o como cargarlo. Existen diferentes gestores de arranque como son:

1- GRUB

En computación, el **GR**and **U**nified **B**ootloader (**GRUB**) es un gestor de arranque múltiple que se usa comúnmente para iniciar dos o más Sistemas Operativos instalados en un mismo ordenador.

Una de las características más interesantes es que no es necesario instalar una partición nueva o un núcleo nuevo, pudiendo cambiar todos los parámetros en el arranque mediante la Consola de GRUB.

Mientras los gestores de arranque convencionales tienen una tabla de bloques en el disco duro, GRUB es capaz de examinar el sistema de archivos. Actualmente, soporta los siguientes sistemas de archivos:

- ext2/ext3 usado por los sistemas UNIX y su variante GNU/Linux
- ext4 en versiones de pruebas.
- ReiserFS.
- XFS de SGI (aunque puede provocar problemas).
- UFS.
- VFAT, como FAT16 y FAT32 usados por Windows 9.x
- NTFS usado por los sistemas Windows NT (a partir de Windows NT v.3.51).
- JFS de IBM.
- HFS de Apple Inc.

GRUB soporta 14 colores de fondo, que normalmente es negro. Algunas distribuciones de Sistemas Operativos que incluyen GRUB frecuentemente utilizan fondos personalizados con el

logotipo de dicha distribución. Los usuarios de GRUB pueden también hacer sus propios fondos.

Proceso de inicio de GRUB

1. El BIOS busca un dispositivo de inicio (como el disco duro) y pasa el control al registro maestro de inicio (Master Boot Record, MBR, los primeros 512 bytes del disco duro).
2. El MBR contiene la fase 1 de GRUB. Como el MBR es pequeño (512 bytes), la fase 1 sólo carga la siguiente fase del GRUB (ubicado físicamente en cualquier parte del disco duro). La fase 1 puede cargar ya sea la fase 1.5 o directamente la 2
3. GRUB fase 1.5 está ubicada en los siguientes 30 kilobytes del disco duro. La fase 1.5 carga la fase 2.
4. GRUB fase 2 (cargada por las fases 1 o 1.5) recibe el control, y presenta al usuario el menú de inicio de GRUB.
5. GRUB carga el kernel seleccionado por el usuario en la memoria y le pasa el control.

2- LILO

Lilo ("**Li**nux **Lo**ader") es un gestor de arranque de Linux que permite iniciar este sistema operativo junto con otras plataformas en el mismo ordenador. LILO funciona en una variedad de sistemas de archivos y puede arrancar un sistema operativo desde el disco duro o desde un disco flexible externo, también permite seleccionar entre 16 imágenes en el arranque y puede instalarse también en el master boot record (MBR).

Al iniciar el sistema LILO solamente puede acceder a los drivers de la BIOS para acceder al disco duro. Por esta razón en BIOS antiguas el área de acceso está limitada a los cilindros numerados de 0 a 1023 de los dos primeros discos duros. En BIOS posteriores LILO puede utilizar sistemas de acceso de 32 bits permitiéndole acceder a toda el área del disco duro.

En las primeras distribuciones de Linux LILO era el gestor de facto utilizado para arrancar el sistema.

3- SYSLINUX.

El Proyecto **SYSLINUX** abarca un conjunto de gestores de arranque ligeros, para arrancar ordenadores en el sistema operativo Linux. Está formado por varios sistemas distintos.

- El **SYSLINUX** original, usado para arrancar desde sistemas de archivos FAT (normalmente discos flexibles).
- **ISOLINUX**, usado para arrancar desde sistemas de archivos ISO 9660 CD-ROM.
- **PXELINUX**, usado para arrancar desde un servidor de red con el sistema Entorno de ejecución de Pre-arranque (PXE).
- **EXTLINUX**, usado para arrancar desde los sistemas de archivos de Linux ext2/ext3.
- **MEMDISK**, usado para arrancar Sistemas Operativos más antiguos como MS-DOS desde memoria.
- Dos sistemas de menús separados.
- Un entorno de desarrollo para módulos adicionales.

SYSLINUX e ISOLINUX

SYSLINUX no se usa normalmente para arrancar instalaciones de Linux completas ya que Linux no suele instalarse en sistemas de archivos FAT. En cambio, se utiliza con frecuencia para discos flexibles de arranque o de rescate, LiveUSBs, u otros sistemas de arranque ligeros. ISOLINUX se utiliza generalmente para LiveCDs de Linux o CDs de arranque instalables.

PXELINUX

PXELINUX se utiliza en conjunción con una imagen adecuada para PXE en una tarjeta de red. El entorno PXE utiliza DHCP o BOOTP para habilitar una conexión TCP/IP básica, y luego descarga un programa de arranque por TFTP. Este programa de arranque carga y configura el kernel de acuerdo a directivas que son obtenidas también desde el servidor TFTP.

Típicamente, PXELINUX se usa para instalaciones de Linux desde un servidor de red central o para arrancar estaciones de trabajo sin disco.

EXTLINUX

EXTLINUX es usado normalmente como un cargador de arranque de propósito general, de forma similar a LILO o GRUB.

3.5 Proceso de arranque de Linux

1. La BIOS.

BIOS TIPO PLCC EN SOCKET REMOVIBLE

BIOS TIPO PLCC SOLDADA EN MAINBOARD

Bios tipo DIP, también las encuentras en socket y soldadas.

Este proceso es común en todos los equipos, así que ya sea un arranque con Linux o con otro sistema operativo este paso será común para todos ellos. El Bios (Basic Input Output System) es el sistema básico de entrada/salida que manipula el proceso de arranque inicial de un ordenador. El código de este programa se encuentra almacenado en una memoria Flash (antiguamente se usaban memorias ROM) que se encuentra instalada en la placa base del ordenador, que puede ser reescrita, y esto permite que se puedan actualizar fácilmente. Además el BIOS se apoya de otra memoria, del tipo CMOS (Complementary Metal-Oxide Semiconductor – Semiconductor de óxido-metal complementario) en ella se cargan y guardan los valores que necesita y que son susceptible de ser modificados (Hora del Sistema, Número de discos duros, secuencia de arranque o inicialización de los dispositivos, configuración de puertos, etc). El proceso de arranque de un ordenador comienza al pulsar el botón de encendido de nuestro equipo otro caso sería utilizar el arranque mediante el sistema Wake On Lan (WOL) integrado en nuestra tarjeta Ethernet. Al activar el mecanismo del pulsador o el envío de un señal WOL se manda una señal eléctrica que indica a la placa base que comienza

con el proceso de arranque. El primer paso será enviar la señal (PS_ON#) para que arranque la fuente de alimentación, una vez se estabiliza la tensión de la fuente, se envía la señal de (PWR_OK) y posteriormente se realizará la carga del programa BIOS almacenado en nuestra placa base. El primero paso es ejecutar la primera instrucción del "Auto diagnostico al encender" conocido como POST (Power-On Self Test). Su primera tarea será la de comprobar, configurar y diagnosticar los componentes conectados en la placa base. El orden en el que se hace esta tarea puede variar según el fabricante del BIOS. Podríamos resumir está secuencia como sigue:

• Verificación del registro del procesador.

• Comprobación del temporizador

• Verificar el BIOS.

• Verificar la configuración del CMOS.

• Inicializar el controlador DMA.

• Verificar e Iniciar la memoria RAM y memoria caché.

• Inicializar los dispositivos de vídeo y teclado.

• Instalar todas las funciones del BIOS.

• Comprobación e Inicio de todas las configuraciones de los periféricos conectados a la placa base (HDD, DVD, etc).

Si en este proceso se encuentra algún error, el POST nos avisará mediante un mensaje en pantalla o mediante avisos sonoros que podremos descifrar junto con el manual del fabricante donde vienen detallados los errores y el significado de los avisos. Si no hay problemas, el BIOS emitirá un sonido corto para comunicar que el proceso del POST se ha realizado con éxito sin errores. En un ordenador hay más de un BIOS, muchos de los componentes del ordenador traen sus propias BIOS, con un programa (firmware) que mejora la gestión del componente, ya que el BIOS de la placa base carga los controladores necesarios, para establecer la primera comunicación con el resto de dispositivos para poder acceder a sus BIOS propias y inicializar los dispositivos con todas sus funcionalidades. Algunas de estos dispositivos pueden ser:

• El BIOS del teclado.

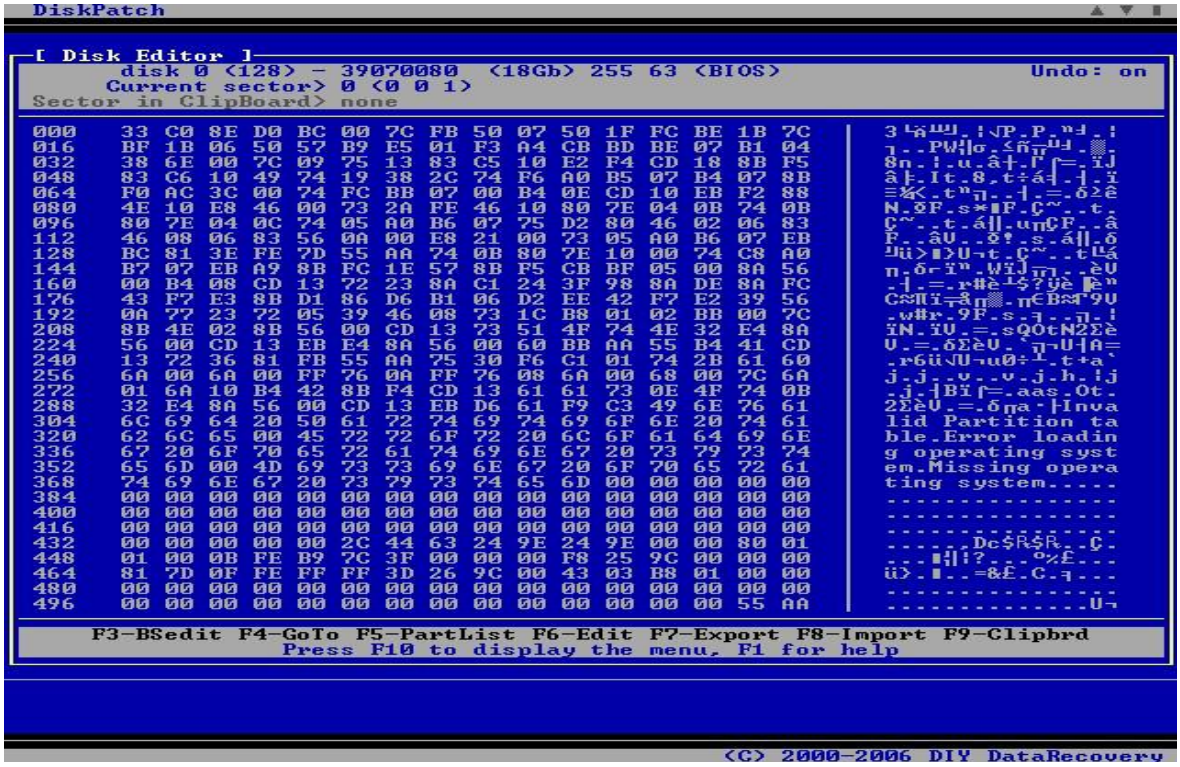
• El BIOS de la tarjeta de vídeo.

• El BIOS de un controlador SCSI.

• El BIOS de una tarjeta de expansión (Firewire, eSATA, USB, etc).

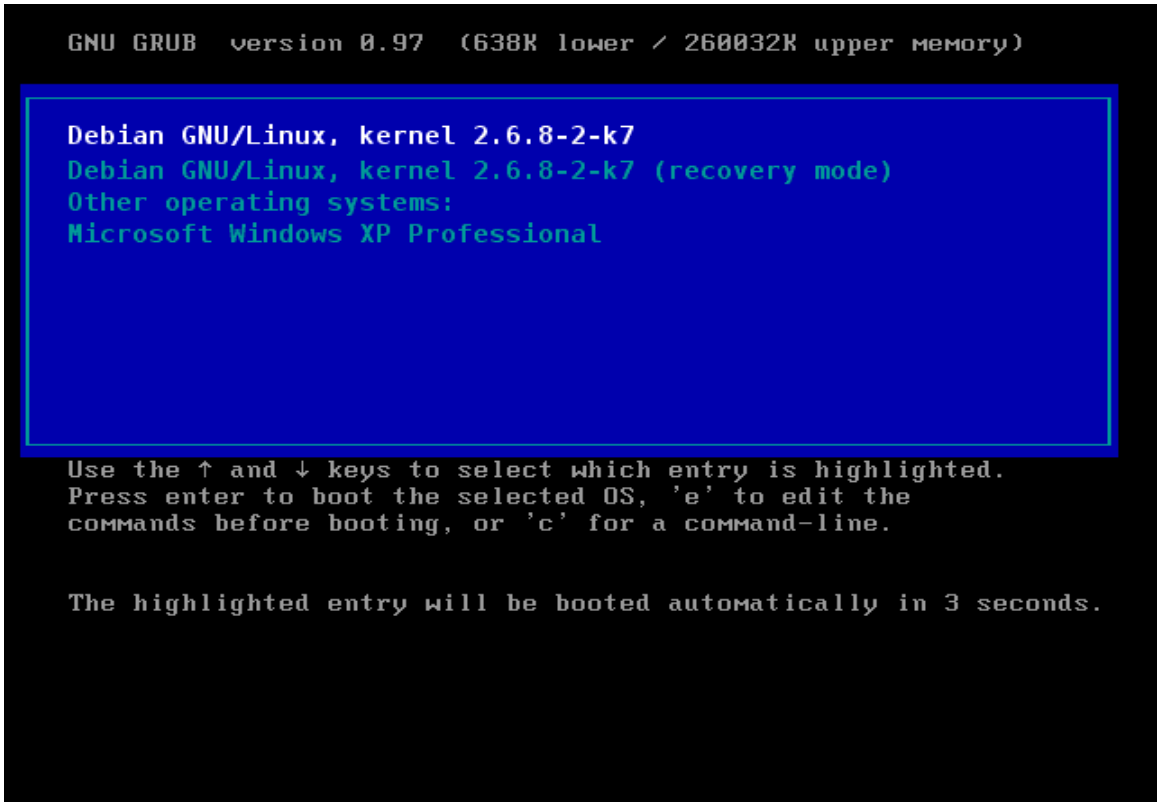
Todos los BIOS generalmente se pueden actualizar siempre que el fabricante saque nuevas versiones de la misma. Con esto se consigue corregir errores, incluir nuevas especificaciones y mejorar el rendimiento del sistema. Antes de finalizar la secuencia POST se comprueba si se ha pulsado la tecla Supr o F2 (generalmente) para finalizar la carga del POST o en caso de que se haya pulsado para iniciar el programa de configuración de los parámetros del BIOS. Finalmente, se comprueba ordenadamente (según la secuencia de arranque) entre todos los dispositivos de almacenamiento, un registro de arranque valido conocido como Registro Maestro de Arranque MBR (Master Boot Record).

2. MBR (Master Boot Record).



El MBR es el primer sector de un dispositivo de almacenamiento (Discos duros, DVD,...) concretamente en el (cilindro 0, cabeza 0, sector 1), el tamaño de este sector es de 512 bytes. Se comprueba si existe un código firmado -valido- con 55H, AAH en los bytes 511 y 512. Se carga las instrucciones de código máquina para arrancar el equipo. En este caso se carga el código modificado por gestor de arranque en memoria, el cual toma el control del arranque y empieza la carga del sistema.

3. Gestor de arranque (GRUB).



Dependiendo de la arquitectura el proceso de carga del sistema operativo diferirá ligeramente.

1. Cargador de arranque básico.

Un cargador de arranque es un programa sencillo que realiza las funciones básicas para poder cargar el sistema operativo. En los ordenadores modernos, normalmente se subdividen en cargadores de varias etapas. El proceso de arranque comienza con la CPU ejecutando los

programas contenidos en la memoria ROM en una dirección predefinida (se configura la CPU para ejecutar este programa, sin ayuda externa, al encender el ordenador). La primera etapa del gestor de arranque, (un código máquina pequeño) normalmente se encuentra alojada en el MBR, y es ésta la que se encarga de cargar el resto del gestor de arranque en memoria.

2. Cargador de arranque de segunda etapa.

Luego se le da paso a los cargadores de segunda etapa, como ejemplo tenemos LILO (más antiguo), GRUB, SILO, NTLDR, SYSLINUX que son los más usados, entre los usuarios de Sistemas Operativos GNU/Linux. Son programas que están limitados en cuanto a operatividad y diseñados exclusivamente para preparar todos los recursos que el sistema operativo necesita para poder funcionar correctamente. El gestor de arranque por defecto suele ser GRUB, tiene la ventaja de leer particiones ext2 y ext3 y cargar su archivo de configuración (/boot/grub/ grub.conf). Con LILO, la segunda etapa es usar la información del MBR para determinar cuales son las opciones de arranque disponibles. Por lo que cuando se actualice el kernel de forma manual deberá de ejecutarse el comando /sbin/lilo -v -v para que la información del MBR sea actualizada. Cuando la primera etapa del gestor de arranque ha conseguido cargar el resto del mismo en memoria, y ha leído del MBR cuales son las particiones arrancables (o que contienen un sistema operativo) el gestor de arranque muestra en pantalla al usuario un menú con todos los Sistemas Operativos que ha encontrado. Puede tener definida, una partición (Sistemas Operativos o kernels) para arrancar en ella por defecto después de un cierto tiempo si el usuario no hace una elección. Puede también configurarse el tiempo de espera, así como un esquema de colores para el menú, opciones de protección por contraseña, etc. Todos estos parámetros se definen en el fichero /boot/ grub/menu.lst (siempre que hablemos de un gestor de arranque GRUB). En éste punto el sistema está preparado para la interacción con el usuario, pudiendo éste elegir el sistema operativo que desea arrancar con las flechas direccionales del teclado.

4. El kernel.

Después de que el usuario elija el sistema operativo, (para el caso en concreto de éste documento sería algún sistema Unix) se carga el kernel del sistema. El kernel del sistema se encarga de los principales procesos del sistema operativo, manejo de memoria, disco, hardware, planificación y comunicación entre procesos, etc. En el proceso del kernel hay dos etapas diferenciables: la carga y la ejecución. El kernel se encuentra comprimido en un archivo,

que se descomprime y carga en memoria, así como los drivers necesarios para que pueda funcionar el hardware del equipo, los cuales se encuentran en el disco RAM (o initrd). Una vez que todo se haya cargado en memoria, se procede a la ejecución. La ejecución empieza con la llamada a la función startup() mediante la cual se maneja toda la memoria (paginación, etc), luego detecta la CPU y sus funcionalidades y posteriormente cambia a funcionalidades independientes del hardware con la llamada a la función start_kernel(). Durante el proceso se monta el disco RAM (que se montó anteriormente como un sistema de archivos temporal, que posteriormente se desmonta durante la función pivot_root() y lo reemplaza por el sistema de archivos real quedando completamente disponible. Cuando el manejo de memoria y la planificación de tareas están listas, el sistema es completamente operacional a nivel de procesos, ejecutando a continuación los procesos init para configurar así el entorno de usuario.

5. Programa init.

El INIT procede consulta un fichero de configuración a nivel de ejecución del sistema, para lo que mira su fichero de configuración, el INITTAB que se encuentra en /etc. Para ello utiliza los RunLevel's, y existen 6 posibles tipos que se identifican por un número:

1. 0 Apagado del sistema

2. 1 Monousuario sin entorno gráfico, sin entorno de red

3. 2 Multiusuario sin entorno gráfico, sin entorno de red

4. 3 Multiusuario sin entorno gráfico pero con entorno de red

5. 4 No se usa por razones históricas

6. 5 Por defecto, Multiusuario, con entorno gráfico, con red

7. 6 Reinicio del sistema

Por ejemplo, si nosotros introducimos en consola "init 0" el sistema se apagaría.

Bueno, ahora INIT hace básicamente dos cosas:

1. Ejecuta scripts de configuración global del sistema rc.sysinit (se encuentra en /etc/rc.d):

- Crea las variables de entorno del sistema
- Activa la partición swap
- Inicializa el reloj
- Controla/chequea el sistema de ficheros ext2/3

2. En función del número de RunLevel se va al directorio /etc/rc.d/rcn.d (para el runlevel 5 seria /etc/rc.d/rc5.d) y allí ejecuta

- todos los scripts que hay dentro:
- kn nombre_proceso –> kill = parar o matar
- sn nombre_proceso –> start = empezar
- A los procesos llamados desde INIT (/etc/rc.d/rcn.d) con los scripts **sn nombre_proceso** se les llama demonios (estos procesos suelen estar en segundo plano ejecutándose continuamente).

Es también el encargado de la adopción de procesos huérfanos que son aquellos cuyo proceso padre murió, puesto que los procesos deben estar en un árbol individual.

3.6 Conclusiones

El uso de las herramientas anteriormente explicadas fue de gran importancia, ya que permitió tener un mejor control y aseguramiento del sistema embebido. Se explicó como fue el proceso de creación de la distribución, así como el gestor de arranque utilizado y además se mostró como es el proceso de arranque de un kernel de linux en un dispositivo, lo cual era necesario dominar.

Capítulo

4

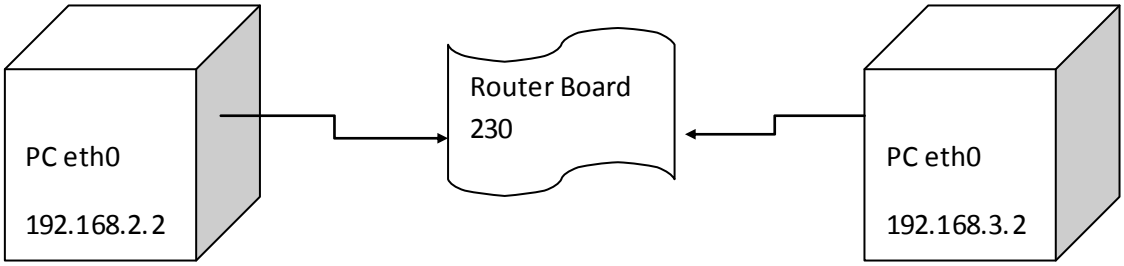
4-Desarrollo de pruebas de laboratorio.

4.1 Introducción

El objetivo de esta etapa del trabajo es realizar una comprobación de la eficacia y eficiencia de nuestra solución. Para lograrlo diseñamos un escenario donde dos PCs son defendidas por el sistema embebido y transfieren la información de forma protegida a través del dispositivo.

4.2 Pruebas desarrolladas

Se simuló una comunicación entre dos computadoras haciendo referencia a dos subredes, las cuales tenían como Gateway o Router a la Routerboard 230. El dispositivo contaba con la incorporación de la distribución NovaRouter y el kernel tenía incorporado el módulo del Sistema de Protección Perimetral.



Posteriormente fue sometido el sistema a las siguientes pruebas:

PING: Utilizamos el comando **ping** para verificar la conectividad entre máquinas de diferentes subredes y los tiempos de respuestas de los mismos. Se utilizó la opción de este comando de enviar paquetes de diferentes longitudes para forzar fragmentación.

Estas pruebas se realizaron con una variedad de kernel demostrando la flexibilidad y compatibilidad del sistema embebido, entre estos kernel están los siguientes:

kernel versión 2.6.16.1

kernel versión 2.6.25.5

kernel versión 2.6.28.2

Finalizadas estas pruebas se obtuvieron resultados positivos y óptimos, se observó que el sistema embebido cumple con su funcionalidad, con una utilización de memoria y capacidad de procesamiento baja.

4.3 Conclusiones

En este capítulo se demostró la importancia que tienen las pruebas para obtener buenos resultados en el proceso de desarrollo de un producto, se explicaron en que consistían las pruebas a las que se sometió el sistema embebido y como fueron los resultados obtenidos.

Conclusiones Generales

Debido a la necesidad de asegurar y proteger el flujo de información que circula por las redes del MININT de una forma óptima, se llevó a cabo un estudio de los sistemas embebidos existentes en el mundo. En este trabajo de diploma se logró elaborar un sistema embebido, controlado por una distribución basada en GNU/Linux, lo que permitió obtener un sistema flexible y seguro, dejando así cumplidos los objetivos trazados en este trabajo de diploma.

También se puede concluir que la herramienta Buildroot utilizada para la creación entornos de compilación cruzada, fue efectiva y ventajosa, ya que nos proporcionó una serie de funcionalidades que optimizaron el proceso de desarrollo de la distribución propia.

Recomendaciones

- Se recomienda hacer un estudio del kernel de linux para aplicaciones en tiempo real, ejemplo: RTLinux, con el objetivo de una posterior optimización del sistema embebido.
- Mejorar la herramienta Buildroot, para obtener más funcionalidades.
- Tomar como punto de apoyo y guía de estudio el presente trabajo de diploma, si se quisiera investigar o desarrollar algo relacionado con los sistemas embebidos.

Bibliografía Referenciada

1. Debian -- Acerca de Debian. [cited 10 Febrero 2009]. Available from World Wide Web: <http://www.debian.org/intro/about>.

2. GNU/Linux embebido: Tesis de grado. [cited 17 Febrero 2009]. Available from World Wide Web: <http://linuxemb.wikidot.com/tesis>.

3. La Definición de Software Libre - Proyecto GNU - Free Software Foundation (FSF). [cited 10 Febrero 2009]. Available from World Wide Web: <http://www.gnu.org/philosophy/free-sw.es.html>.

4. RouterBOARD 200 Series
User's Manual. [cited 13 Febrero 2009]. Available from World Wide Web: <http://www.routerboard.com/pdf/RouterBOARD200SeriesA4-3-0.pdf>.

5. Routerboard.com. [cited 17 Febrero 2009]. Available from World Wide Web: <http://www.routerboard.com/rb200.html>.

6. Ubuntu - doc.ubuntu-es. [cited 10 Febrero 2009]. Available from World Wide Web: <http://doc.ubuntu-es.org/Sobre_Ubuntu>.

7. http://es.asus.com/products.aspx?l1=3&l2=116&l3=0&l4=0&model=1146&modelmenu=1

8. José Antonio Vega Hermosilla, Yosley Bello Rodríguez. *Desarrollo de una solución propia para el enrutamiento sobre GNU/Linux y su interfaz de administración remota*.

9. Richard Stallman. Linux y GNU - Proyecto GNU - Fundación para el Software Libre (FSF). [cited 10 Febrero 2009]. Available from World Wide Web: <http://www.gnu.org/gnu/linux-and-gnu.es.html>.

Bibliografía Consultada

1. Building Embedded Linux Systems - Búsqueda de libros de Google. [cited 10 Febrero 2009]. Available from World Wide Web: <http://books.google.com.cu/books?id=I-hVqkX6A9cC&printsec=frontcover&dq=Building+Embedded+Linux+Systems#PPA370,M1>.

2. Compunauta Micro Linux (uLinux) - FREE MicroLinux II. [cited 14 Febrero 2009]. Available from World Wide Web: <http://www.compunauta.com/uLinux/>.

3. Embedded System
   Tools Guide. Junio 2004.

4. ROUTERBOARD 230 COMPATIBLE CON TODOS LOS SISTEMAS OPERATIVOS - 3897209 | MasOportunidades.com.ar. [cited 13 Febrero 2009]. Available from World Wide Web: <http://www.masoportunidades.com.ar/aviso/3897209-routerboard-230-compatible-con-todos-los-sistemas-operativos>.

5. ThinLinux.org. [cited 14 Febrero 2009]. Available from World Wide Web: <https://linuxdevices.com/links/LK2967815104.html>.

6. Alessio Bechini, Cosimo Antonio Prete. *Support for Architectural Design and Re-Design of Embedded Systems*.

7. AlMarzouq, Mohammand. Free and Open Source Software. [cited 9 Febrero 2009]. Available from World Wide Web: <http://www.infosci-online.com/downloadPDF/encyclopedias/IGR25445_DYCMdWf8OP.pdf>.

8. Bruce Perens. Building Tiny Linux Systems with Busybox--Part I. [cited 10 Febrero 2009]. Available from World Wide Web: <http://www.linuxjournal.com/node/4335/print>.

9. Darrick Addison . Embedded Linux applications: An overview. [cited 14 Febrero 2009]. Available from world wide web: <http://www.ibm.com/developerworks/linux/library/l-embl.html?l=sd,t=gr,p=emblinx>.

10. [cited 9 Febrero 2009]. Available from World Wide Web: <http://www.infosci-online.com/downloadPDF/pdf/ITB10127_3NhRJM1O0S.pdf>.

11. Luis Germán G. Sistemas Embebidos. [cited 11 Febrero 2009]. Available from World Wide Web: <http://electronica.udea.edu.co/cursos/sistembebidos.htm>.

12. Stuart Lynne. ThinLinux Guide. [cited 14 Febrero 2009]. Available from World Wide Web: <http://linux-int.carnet.hr/knjige/ThinLinux_20Guide1.pdf>.

13. Tere Vadén, Niklas Vainio. Free Software Philosophy and Open Source. [cited 9 Febrero 2009]. Available from World Wide Web: <http://www.infosci-online.com/downloadPDF/encyclopedias/IGR3324_V4FH2M7GVA.pdf>.

14. Tony Mancill. Linux Systems as Network Infrastructure Components. [cited 9 Febrero 2009]. Available from World Wide Web: <http://www.infosci-online.com/downloadPDF/pdf/ITB7361_8dKGC2DEUN.pdf>.

15. http://es.wikipedia.org/wiki/BIOS
16. http://es.wikipedia.org/wiki/POST
17. http://www.intel.com/support/mt/sp/motherboards/desktop/sb/cs-025434.htm
18. https://docs.google.com/Doc?id=dhh373x4_5d4hvdrhs&hl=es
19. http://www.tu-chemnitz.de/docs/lindocs/RH9/RH-DOCS/rhl-rg-es-9/s1-bootinit-shutdown-process.html
20. Guía Para Administradores de Sistemas GNU/Linux Versión 0.8
21. Proceso de arranque y carga de linux http://albertjh.cymaho.com/?p=275