Cortafuegos (Firewall) basado en Microcontrolador

ALFREDO ADRIÁN ORTEGA

Tesina presentada para el grado de
Licenciado en Informática

Departamento de Informática
Facultad de Ingeniería
Universidad Nacional de la Patagonia San Juan Bosco

Trelew - Chubut
Argentina

Junio del 2003

Cortafuegos(firewall) basado en Microcontrolador

ALFREDO ADRIÁN ORTEGA

Tesina enviada al Departamento de Informática como requerimiento parcial para el grado de Licenciado en Informática
Junio del 2003

Resumen

Los sistemas de cortafuegos son ampliamente utilizados hoy en día para proteger todo tipo de redes de accesos no autorizados, sin embargo, este servicio se incluye generalmente dentro del sistema operativo del servidor, y no es fácilmente configurable por el usuario final, quien puede tener otras necesidades de protección. Otra desventaja de los cortafuegos típicos es que son propensos a cortes del servicio cuando el hardware del servidor falla.

Se propone realizar un sistema de cortafuegos que sea portable desde el punto de vista físico, y fácilmente configurable, que cumpliría el rol de cortafuegos personal para quien necesite seguridad y movilidad al mismo tiempo.

Área o campo de aplicación

Hardware y Software de Microcontroladores - Arquitectura de Computadoras - Interfases de Entrada/Salida - Sistemas Operativos - Protocolos de Red Local (Ethernet) - Protocolos de Internet (TCP/IP) - Seguridad en Redes

Palabras Clave: Microcontrolador, Firewall

Tutor:

ING. ALEJANDRO ROSALES

Profesor Asociado de Arquitectura de los Sistemas de Cómputo

Profesor Adjunto de Redes y Transmisión de Datos

Facultad de Ingeniería - Sede Trelew

Correo: arosales@tw.unp.edu.ar

Colaborador en Tutoría:

ING. FABIO SALERNO

Jefe de Trabajos Prácticos de Redes y Transmisión de Datos

Auxiliar de 1ra de Sistemas Distribuidos

Facultad de Ingeniería - Sede Trelew

Declaración

Esta tesina está basada en investigación y desarrollo llevada a cabo por el Autor. Ninguna parte de esta tesis ha sido enviada a otro lugar para la obtención de otro grado o calificación y fue realizada por el Autor excepto que se especifique lo contrario en el texto.

Reconocimientos

Se agradece al personal del Departamento de Física y el Departamento de Sistemas de la Facultad de Ingeniería, por haber prestado sus equipos y su tiempo para la realización y prueba de esta tesina.

Al ING. ARIEL YUVONE de Servycom por haber importado componentes necesarios para la construcción del dispositivo.

A mi hermano J. M. M. ORTEGA por haber diseñado gran parte del circuito impreso y colaborado en la confección del mismo.

A mis hermanas EUGENIA y MARÍA por dejarme sacrificar su portarretratos acrílico como soporte físico para el dispositivo.

A mis padres por su constante aliento.

A la Nación Argentina por haberme posibilitado estudiar una carrera Universitaria en forma gratuita.

Índice general

Índice de figuras

Listados

1. Introducción

1.1. Motivación

Existe actualmente un auge por la interconexión de redes, y todo tipo de software ha surgido para incrementar la seguridad de las mismas, pero la mayoría de este software necesita de un equipo estándar (Léase: PC o equipos mayores) que son relativamente costosos y no están diseñados para ambientes industriales o de alta movilidad.

1.2. Objetivos

Se propone el desarrollo de un dispositivo de red de bajo costo, bajo consumo y robustez mecánica, que tenga la capacidad de monitorear las comunicaciones de un equipo o sub-red, ubicado como se ve en la figura 1.1. A la vez, es conveniente que sea lo suficientemente pequeño como para poder transportarse fácilmente, o ubicarse en compartimientos estancos o protegidos, aptos para su uso industrial[1].

Desde el punto de vista del software, debería tratarse de una solución adaptable y flexible, utilizando técnicas modernas de ingeniería de software para facilitar la tarea al programador, y evitar realizar una solución monolítica, que se aplica clásicamente a este tipo de desarrollo. Como se ve en el diagrama superior, la utilización típica de sistema sería entre una PC y una red insegura, aunque pueden existir redes en ambos extremos.

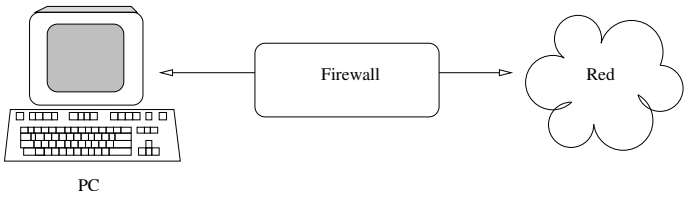


Figura 1.1.: Diagrama General

[1] Los dispositivos usados comúnmente para control industrial (Ej. PLC), no son aptos para el uso personal ya que los modelos económicos no disponen del poder de procesamiento ni las interfaces necesarias para este tipo de tarea, aunque esto puede cambiar con el avance de la tecnología.

1.3. Fundamentación

El hardware del firewall utiliza un dispositivo microcontrolador[2], debido a su bajo precio y alta capacidad de cálculo que han alcanzado recientemente. Para la realización del proyecto se utiliza una amplia variedad de conocimientos del área de la informática, entre ellos:

- Arquitectura de Sistemas
- Redes
- Sistemas Operativos
- Ingeniería de Software
- Lenguajes de Programación

Existen algunos conocimientos necesarios que no se adquieren durante la carrera de Licenciatura, específicamente los relacionados con el diseño del hardware y su configuración eléctrica y electrónica. El autor adquirió esos conocimientos anteriormente[3].

Se considera que es importante el aporte tecnológico que este trabajo realizaría a la región, ya que actualmente se debe importar todo este tipo de tecnología, y aunque los insumos de hardware utilizados son de hecho importados, el costo total es mucho menor que un dispositivo similar fabricado en el exterior, ya que el diseño completo y armado se realiza en el país. Como resultado del tipo de cambio actual, la producción sería viable económicamente.

La búsqueda de información se realizó principalmente mediante Internet. Utilizando buscadores se ubicaron varios sitios que brindan información y herramientas. No se considera a estas investigaciones como una parte importante del trabajo ya que mucha información puede extraerse directamente de la bibliografía utilizada en la carrera de Licenciatura. Las herramientas de programación también proveen ejemplos y ayudas adicionales.

Como puede verse en la figura 1.2, se pensaron distintas formas de implementar un firewall, desde el punto de vista de las interfaces de comunicación. La primera propuesta, de intercalar el sistema directamente en línea telefónica entre un modem y una PC fue rápidamente descartada debido a la lentitud propia de la interfaz, y principalmente por la imposibilidad técnica de manipular las señales analógicas producidas por el modem.

Una segunda propuesta utiliza al sistema como un conversor USB/Ethernet. La idea no es nueva y existen dispositivos que cumplen funciones similares, pero tiene desventajas tales como la lentitud de la interfaz USB (Versión 1.0) con respecto a cualquier versión de Ethernet, y la necesidad de realizar drivers específicos necesarios para que el sistema operativo de la PC reconozca al dispositivo USB como una interfaz de red.

La ultima propuesta fue la que se realizó, y muestra un dispositivo completamente independiente que consta de dos interfaces Ethernet, capaces de conectarse directamente

[2] La diferencia de un Microcontrolador (MCU o Micro Controller Unit) con un CPU (Central Process Unit) es que este ultimo sólo realiza el procesamiento, mientras que un MCU generalmente incluye memoria, CPU y periféricos en un mismo circuito integrado.

[3] En la E.N.E.T. Nº1 de Trelew, -(título de Técnico Electrónico con Orientación Digital.)-

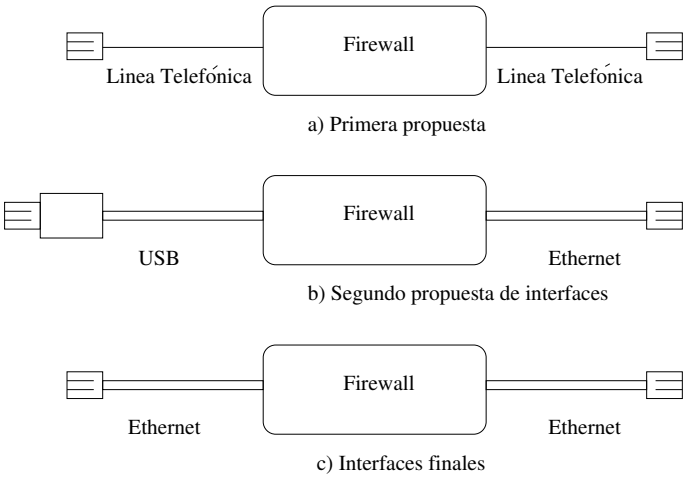


Figura 1.2.: Distintas Configuraciones

a un computador, o a otro dispositivo de red tal como un hub o switch, pudiendose utilizar efectivamente como un firewall entre redes.

1.4. Metodología y recursos necesarios

La metodología y recursos necesarios para el desarrollo son distintas en el caso del hardware que del software, ya que los niveles de complejidad y conocimiento son muy distintos

1.4.1. Desarrollo del Hardware

Metodología

Se utilizó un enfoque cerrado de creación de prototipos, o sea, primeramente se crea un *prototipo desechable*[4], en el que se demuestra la viabilidad del diseño (Pressman:1997). Este prototipo se descarta[5] y se procede al diseño final propiamente dicho.

Recursos

Los recursos necesarios para la fabricación del hardware son pocos, más allá de los componentes mismos. Se necesitó un multi-tester para probar las conexiones eléctricas y los implementos químicos necesarios para la confección del circuito impreso. Todos los insumos y componentes fueron adquiridos dentro del país, a excepción de las interfaces de red, que tuvieron que importarse especialmente.

El diseño del circuito impreso fue realizado mediante un software de CAD muy popular. La información detallada se encuentra en el apéndice C en la página 102.

[4]Se utilizó un PROTO-BOARD, (dispositivo comúnmente utilizado para la creación de prototipos electrónicos).

[5]No se desechó literalmente, sino que se desarmó.

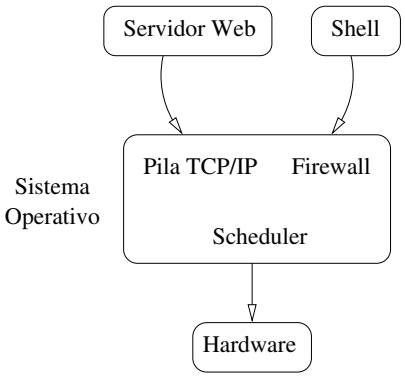


Figura 1.3.: Diseño final del software, se observa la estructura en capas.

1.4.2. Desarrollo del software

Metodología

En este caso se utilizó un enfoque abierto de prototipado, o sea, un *prototipo evolutivo* que no se desecha. En un principio se consideró la idea de utilizar técnicas de desarrollo de software de Tiempo Real, que simula la concurrencia de varios procesos utilizando interrupciones y máquinas de estado, pero luego de observar las capacidades del MCU, se decidió por un enfoque más convencional, utilizando un sistema operativo de tiempo real que provea servicios de multitarea reales, y una aplicación que corra sobre el mismo (figura 1.3).

El termino ’Tiempo Real’ se aplica a muchos sistemas embebidos, una definición aceptada es:

> Un sistema se considera de ’Tiempo Real’, si el *tiempo* es una de las variables que maneja.

Esta definición es un poco general, por lo que suele causar confusiones. También se suele llamar a estos sistemas ’*MicroKernels*’ aludiendo a su pequeño tamaño, termino que se confunde con el utilizado para ciertos tipos de sistemas operativos. Finalmente, también se suelen denominar como ’*Determinísticos*’, aludiendo al hecho que puede predecirse la instrucción exacta que se ejecuta en un determinado momento. Para esto, el sistema operativo debe poseer una latencia constante en todas sus funciones, o de lo contrario no se podría predecir con exactitud el comportamiento del sistema.

El lenguaje de programación escogido fue “ansi C” o “C” estándard. Los motivos se desarrollan en el capítulo 2, pero se puede nombrar aquí la portabilidad y el bajo nivel de este lenguaje como las características que lo adecuan al proyecto.

Recursos

Se necesita un sistema relativamente modesto de desarrollo, a estándares actuales. Pero hacen falta herramientas de software, a saber:

- Un IDE[6]/ensamblador: Generalmente las empresas fabricantes de MCU proveen versiones gratuitas o limitadas de estas herramientas, ya que son indispensables. Para el proyecto, la empresa microchip http://www.microchip.com provee un paquete muy completo que permite trabajar con el código de ensamblador del dispositivo.

- Un compilador de C: Indispensable para cualquier proyecto de envergadura, suelen existir compiladores de C para prácticamente cualquier dispositivo programable. Pero en contraste, no todos son gratuitos, y pueden llegar a ser herramientas muy caras. Durante el proyecto se utilizaron varios compiladores de diferentes empresas, pero se seleccionó el software C18 de Microchip, del cual existen versiones de evaluación gratuitas.

- Capturador/Analizador de Red: Fundamental para analizar el tráfico y detectar errores. Uno de los mejores disponible en forma gratuita es ethereal http://www.ethereal.com.

- Emulador de Terminal: Hyperterminal en entornos Windows, o Minicom, su correspondiente en Linux.

- Otras herramientas de soporte y documentación. Se incluye la lista completa en el apéndice A en la página 98.

Para realizar las pruebas de forma completa, se necesita una red informática real con sistemas heterogéneos, tal como la existente en el Depto. de Física o el Depto. de Informática de la Universidad.

1.5. Plan de actividades y Cronograma

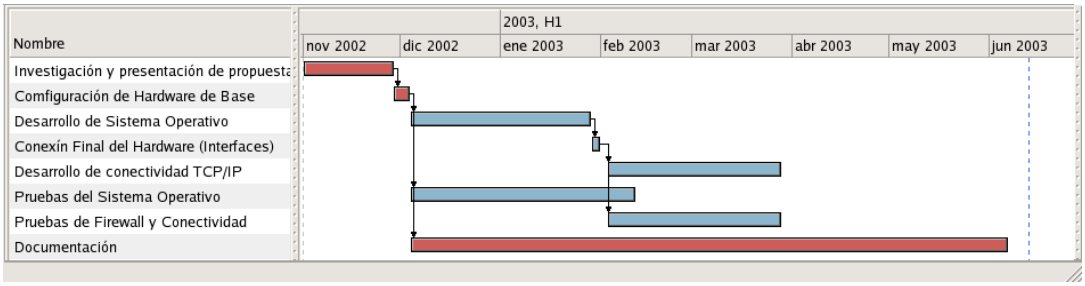


Figura 1.4.: Diagrama de Gantt

[6]Integrated Development Enviroment, Entorno Integrado de Desarrollo, generalmente combina un editor de texto especializado junto con un compilador.

2. Desarrollo del Hardware

2.1. Idea inicial

La idea surge de un práctico realizado para la cátedra de Redes y Transmisión de Datos, que consistió en la construcción de un servidor Web utilizando el microcontrolador PIC16F84, siempre de la empresa Microchip[1]. Se utilizó para ello una versión GPL desarrollada por muchos autores, la cual está disponible en el sitio web http://www.kyllikki.fluff.org/hardware/wwwpic2. wwwpic2 es un servidor WEB que utiliza 800 bytes de FLASH del microcontrolador (aproximadamente), dejando unos 400 bytes libres para almacenar el sitio. La comunicación se realiza mediante una interfaz serie RS-232, utilizando el protocolo SLIP. Esto implica que el servidor no es capaz de funcionar como un dispositivo autónomo, ya que para conectarse a una red local necesita al menos de una PC o router para realizar el encaminado de los paquetes hacia o desde el dispositivo. El hardware es muy simple aunque no lo es tanto el firmware, realizado completamente en código ensamblador. Se debe aclarar que, a diferencia del proyecto actual, la mayor parte del proyecto anterior fue copiado de Internet y modificado para adecuarlo a la cátedra, como lo permite la licencia del mismo. Las modificaciones no fueron pocas y se detallan en el informe presentado para la cátedra.

De este proyecto surgió la idea de la construcción de un firewall, utilizando dispositivos similares pero esta vez, con una interfaz para conectarse directamente a una red local como lo es Ethernet. Inicialmente se había pensado en construir el dispositivo utilizando el procesador PIC16F877 y dos placas de red ISA Ethernet 10Base-T comunes, pero la idea se abandonó, al estar disponible un procesador mucho más moderno como lo es el PIC18F452, al que en adelante se referirá como PIC-18. Se decidió no utilizar placas ISA comunes como interfaces de red debido a su baja disponibilidad actual, ya que son obsoletas y no se consiguen en grandes cantidades. Por otra parte, es posible conseguir módulos integrados que realicen la misma función, mucho más pequeños aunque también mucho más caros.

En la figura 2.1 puede verse el diagrama del nuevo sistema completo, tal como fué presentado.

[1] En palabras de un sitio web dedicado a los Microcontroladores "*Microchip es una empresa que se propone dominar al mundo, vendiendo dispositivos microprogramables accesibles a todo el mundo, y ofreciendo gratuitamente herramientas de desarrollo profesionales*". Aunque el autor no comparte este argumento tan extremo, es cierto que la calidad de sus productos y soporte es muy bueno, y la competencia (Motorola, Ubicom, etc.), aún al poseer dispositivos que hasta cuadriplican la velocidad a precios similares, prácticamente no se dispone de herramientas de desarrollo gratuitas, por lo que su uso se restringe a ambientes industriales, con un presupuesto mucho mayor al disponible.

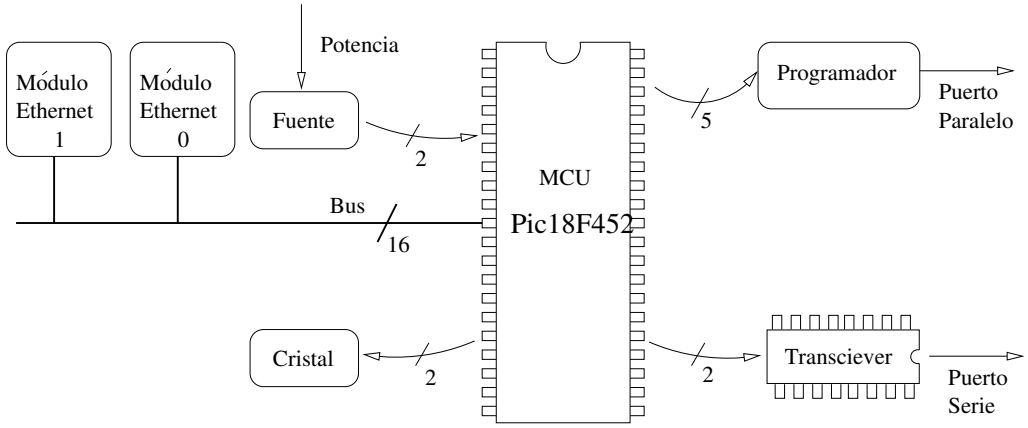


Figura 2.1.: Diagrama simplificado del sistema

2.2. Interfaces de Red

Actualmente la mayoría de las redes locales utilizan la tecnología de Fast-Ethernet[2] a 100 Megabits, switcheada. Esto significa que se utilizan placas de red tipo Ethernet 100, full duplex[3], y la conexión no utiliza hubs (repetidores), sino switchs, equipos que a pesar de tener un mayor costo que un repetidor, contribuyen mucho a aliviar el tráfico total de una red local. Lo importante de todo esto, es que la tecnología Ethernet se ha diseñado teniendo en cuenta la compatibilidad con versiones antiguas, esto es, los dispositivos de red actuales suelen aceptar y autoconfigurarse para distintas velocidades por tramo, sean 10 Megabits, 100 Megabits y hasta 1 Gigabit (por segundo).

La interfaz Ethernet en su versión de 10 Megabits por UTP denominada 10 Base-T, utiliza un par trenzado para la señal saliente y otro para la señal entrante. Dicha señal es codificada en banda base utilizando el código Manchester. Este código es muy útil para sincronización, pero tiene la desventaja de aumentar el ancho de banda de la señal al doble, o sea que por un cable que transporta señales de Ethernet 10 Base-T circula efectivamente una señal de 20 Mhz, que tiene una amplitud de unos 5 Volts a la salida pero se permite gran atenuación por el cable, pudiéndose utilizar a distancias de hasta 100 metros entre nodos.

2.2.1. Problema del Consumo

La interfaz Ethernet no es una interfaz diseñada para el bajo consumo de energía. Tal es así que necesitan unos 100 Mili-Amperes de corriente para funcionar, cada una. Como comparación, el microcontrolador PIC-18, a la máxima velocidad y potencia, consume 10 veces menos. Todo el sistema consume aproximadamente 1 Watt (dos interfaces Ethernet+microcontrolador), potencia irrisoria comparada con una CPU *Pentium 4* de 70

[2]Excepto en los lugares donde se realizaron contratos con IBM, donde se sigue utilizando redes del tipo “Token Ring”, de características similares en cuanto rendimiento, pero un funcionamiento bastante diferente. ver Stallings:99

[3]Se consiguen por unos 10 u$s, aunque la mayoría de los equipos modernos incluyen una o dos integrada en su placa madre.

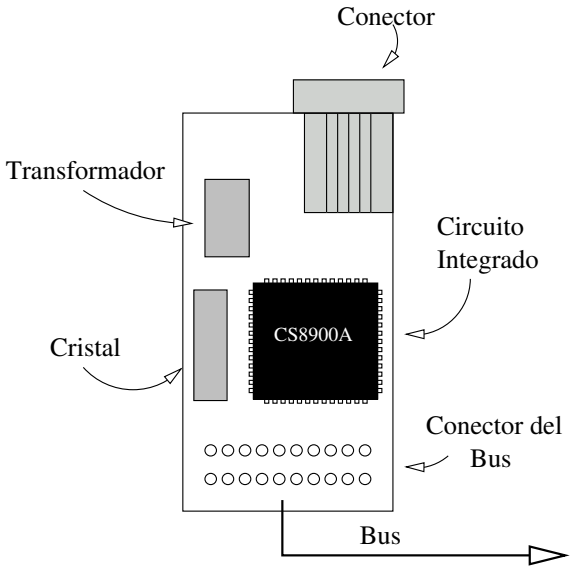


Figura 2.2.: Diagrama del módulo de Red Ethernet

Watts. Aun así, aunque un consumo de un watt para todo el sistema sea relativamente reducido, es demasiado para una batería estándar que no soportará más de un día conectada al dispositivo.

No solamente este proyecto tiene el problema anterior, sino también la mayoría de los dispositivos Ethernet diseñados para utilizarse a la intemperie o a largas distancias de la red local, tales como los routers y antenas inalámbricas (Estandard IEEE 802.11g), que tienen auge en la actualidad. Estos dispositivos necesitan un cable adicional para llevar energía a donde estén ubicados, lo cual complica la instalación de los mismos.

Teniendo en cuenta este problema se está desarrollando un estándar denominado PoE (Power over Ethernet, o potencia a través de Ethernet) en los que un dispositivo compatible con el estándar PoE utiliza un par adicional del cable UTP-5 (El cable tiene 4 pares, Ethernet en casi todas sus versiones utiliza sólo dos) para suministrar potencia a través de largas distancias (ver estándar IEEE 802.3af "*DTE Power via MDI*") . El dispositivo no necesita muchas modificaciones para ser compatible con este estándar, solamente las de llevar los cables de energía hacia la entrada del circuito integrado regulador[4], que acepta voltajes de 7 a 35 Voltios.

2.2.2. Componentes

Una interfaz Ethernet no solamente consta del controlador, sino que necesita una mínima cantidad de componentes externos, que suelen agruparse en una placa de expansión para PC, o un módulo integrado para microcontroladores.

Por Internet se localizó a un comerciante que ofrece módulos Ethernet a 10 Megabits. Dicho módulo se ilustra en la figura 2.2 y consta de:

[4]Una versión SMD del 7805, integrado que frente a un voltaje de 7 a 35 voltios, ofrece una salida de 5 voltios estabilizados y 1 Amper de corriente máxima.

- El circuito integrado CS8900A, de Cirrus Logic, que ofrece un controlador completo de Ethernet a 10 Megabits con memoria interna, configurado por hardware en modo 8 bits.
- Conectores para el bus de datos y direcciones del CS8900A
- Cristal de 20 Mhz
- Transformador aislador. Componente muy difícil de conseguir en el país, se utiliza para aislar los amplificadores del controlador Ethernet de par trenzado UTP.

Se adquirieron dos unidades, que se detallan en el capítulo 6 en la página 63.

2.3. Microcontrolador

El cerebro del sistema es por supuesto el microcontrolador, que concentra una CPU, memoria, periféricos y oscilador en un solo dispositivo de estado sólido.

2.3.1. Selección

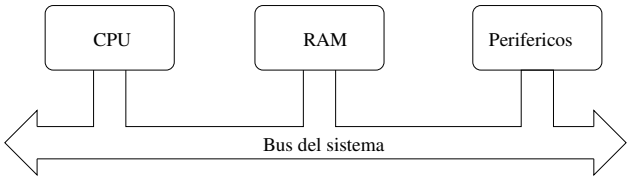
Primeramente se escogió el modelo PIC16F877 de la firma Microchip, pero más tarde se seleccionó otro modelo más avanzado de la misma empresa, el PIC18F452. En la tabla pueden comparase y apreciar el por qué de la selección.

Nombre	Instrucciones	Ram	Rom (Flash)	Velocidad	Precio
PIC16F877	35(14 bits)	368 bytes	8192 Instr. (14 bits)	20 Mhz	9 U$S
PIC18F452	77(16 bits)	1536 bytes	16384 Instr. (16 bits)	40 Mhz	11 U$S

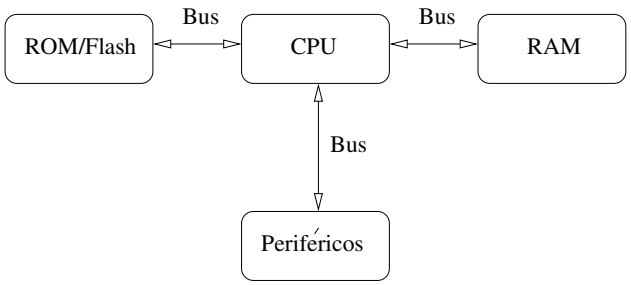
Cuadro 2.2.: Tabla Comparativa entre los modelos seleccionados

Los Microcontroladores actuales son muy sencillos de utilizar, y ha quedado atrás la época en la que se necesitaba un costoso equipo de grabación y borrado UV(Ultra Violeta), o la tecnología OTP (One time programming) que sólo permitía programar al dispositivo una sola vez[5]. Los modernos microcontroladores de 8 bits cuentan con memoria Flash, en una cantidad que varía de 2 a 128 Kbytes. Este tipo de memoria permite reprogramarlos miles de veces durante el ciclo de desarrollo. Durante este proyecto la cantidad de ciclos de borrado/reprogramación se acercaron al millar. La programación se realiza conectando un circuito muy sencillo entre el microcontrolador y la salida paralela o serie de una PC estándar, circuito comúnmente denominado “Programador de Firmware” o simplemente “Programador”.

[5]Estas técnicas aún se utilizan para producción en grandes cantidades, y las empresas siguen ofreciendo estas opciones, o sea, un modelo de microcontrolador suele venir en tres modelos: programación UV (Ultra Violeta), OTP y Flash/EEprom



(a) Von Neumann (x86)



(b) Harvard (Pic)

Figura 2.3.: Diferentes Arquitecturas de sistemas

2.3.2. Arquitectura

Una característica importante de estos microcontroladores es que poseen una arquitectura del tipo Harvard, en contraste de la Von Neumann (UYM:2000). Esta arquitectura permite buses de diferente ancho hacia la memoria de datos (RAM) y hacia la memoria de instrucciones (ROM o Flash), como puede verse en la Figura 2.3, punto b). Esto, aunque implica mayor complejidad, permite reducir los costos y optimizar el rendimiento al disponer de buses independientes[6]. Desde el punto de vista del programador, es un mundo completamente distinto, por ejemplo:

- No se puede leer ni modificar la memoria de instrucciones sino utilizando instrucciones especiales.
- No se disponen de registros en el sentido de los procesadores x86, ya que no hay una clara distinción entre la memoria RAM y los registros del CPU, al menos desde un punto de vista lógico. Por ejemplo, el registro WREG, utilizado como el acumulador que equivaldría al registro EAX de arquitecturas Intel, en realidad puede leerse como la posición de memoria FE8h (hexadecimal), y cada registro tiene su posición individual. Como contrapartida, no tiene sentido preguntarse la dirección de memoria del registro EAX de un procesador Intel, ya que existe en un espacio de memoria separado del resto de la memoria del sistema.

[6]La Arquitectura de Sistemas actual, se está alejando de la planteada por Von Neumann, para acercarse más a la Harvard, como lo demuestran la multitud de buses dedicados dentro de una placa madre moderna, Ej.: AGP para video, PCI para periféricos, acceso a la memoria por un bus especial DDR, etc.

- No existen instrucciones para acceso indirecto a la memoria (Ej. MOV [EBX],xxx) sino que se debe utilizar un mecanismo escribiendo valores en posiciones de memoria especiales.
- La pila de llamadas a sub-programas utilizada por instrucciones CALL está implementada en hardware, en un espacio de memoria separado de la memoria RAM.

A su vez, existen otras diferencias que surgen debido al tipo de instrucciones utilizadas:

- Un procesador x86 puede clasificarse como CISC (Complex Instruction Set CPU, CPU con un conjunto complejo de instrucciones), el programador tiene disponibles unas 300 instrucciones distintas, que puede utilizar con 15 registros de 32 bits[7], aunque no ortogonalmente, o sea, los registros están especializados y no todos pueden ser usados con todas las instrucciones. Dichas instrucciones tienen un largo variable, desde 1 byte hasta más de 10 bytes, y la cantidad de ciclos de reloj necesarios para ejecutar cada instrucción varía. Al ejecutar el código, el mismo se carga desde el disco rígido o diskette a la memoria RAM y es ejecutado desde allí por el sistema operativo.
- En un procesador PIC, hay disponible un número mucho menor de instrucciones[8] , por eso es su denominación de procesadores RISC (Reduced Instruction Set CPU). La mayoría de las instrucciones tienen una longitud fija, cuyo tamaño en bits puede no ser un múltiplo de 8 (Ej. 14 bits en los modelos PIC16), y generalmente se ejecutan directamente desde la memoria rom/flash en una determinada cantidad fija de ciclos de reloj, permitiendo calcular con precisión la duración de la ejecución de un algoritmo.

Afortunadamente la mayoría de estos detalles se ocultan al utilizar un lenguaje como C, con un mayor nivel de abstracción que el código ensamblador.

2.3.3. Selección del Oscilador

Un componente muy importante de todo CPU actual, es el oscilador o reloj del sistema, que marca la velocidad a la cual de procesan las instrucciones. El modelo de microcontrolador escogido es uno de los más avanzados en este sentido ya que provee muchas opciones, algunas de las cuales son:

- Un oscilador de baja potencia de 32 Khz, para ultra-bajo consumo (en el orden de los micro-Amperes).
- Un oscilador que sólo necesita una resistencia y un capacitor externo, de baja precisión.
- Un oscilador PLL que cuadriplica la frecuencia dada por un cristal externo.

[7] La tecnología MMX introdujo algunos de 64 bits, y luego se sucedieron multitud de nuevas extensiones denominadas SSE, SSE2, 3dNOW!, etc.

[8] 35 en la mayoría de los modelos, sólo la línea PIC-18 expandió el set de instrucciones incluyendo algunas específicas para ser utilizadas mediante un compilador de “C”.

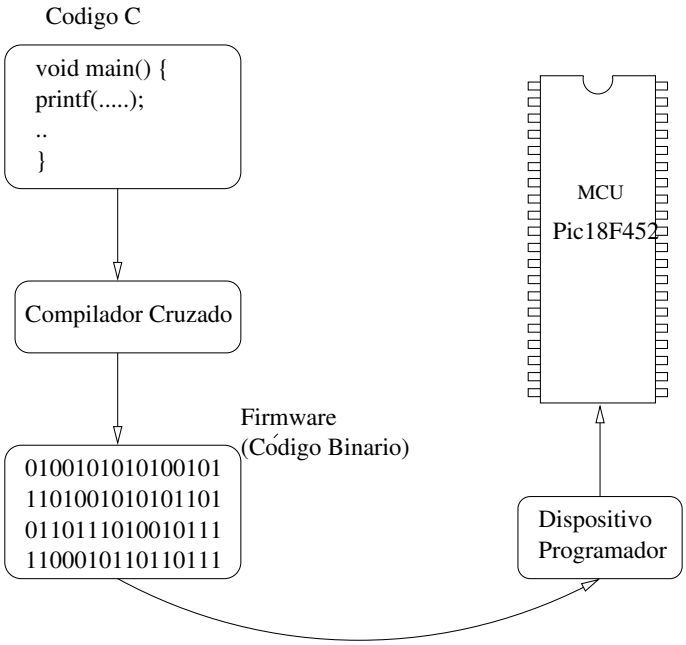


Figura 2.4.: Esquema del ciclo de desarrollo utilizado

La opción utilizada fue la del PLL (Phase Loop Lock, Anclaje de Bucle por Fase), ya que ofrece la facilidad de configuración de un cristal de 10 Mhz que se cuadriplica, obteniendo la velocidad máxima ofrecida por el fabricante, 40 Mhz. Esta tecnología es similar a la ofrecida actualmente por los procesadores modernos, que dada una velocidad del bus relativamente baja (100-400 Mhz) obtienen una oscilación interna del orden de los 2-3 GHz, utilizando multiplicadores. (Ej. el 486DX4 utilizaba una técnica similar para llegar a 100 MHZ de velocidad interna utilizando una velocidad de 25 Mhz de bus).

2.4. Programador de Firmware

2.4.1. Introducción

En esta sección se detalla una parte esencial del proceso de desarrollo de software para microcontroladores, que es el proceso de programar el dispositivo. El código no se denomina “software” como en computadoras normales, ya que este término se utiliza para resaltar la naturaleza transitoria de las instrucciones de todas las computadoras, en las que se puede descargar una aplicación y cargar otra rápidamente en la memoria. El código de un microcontrolador se caracteriza por no cambiar frecuentemente, y es posible afirmar que en la enorme mayoría de las aplicaciones en las que intervienen los microcontroladores, las instrucciones no cambian nunca durante la vida útil del dispositivo, sea porque no es necesario o porque la tecnología utilizada no lo permite (Ej. microcontroladores OTP, sigla de One Time Programming, que sólo se programan una vez). Entonces, el nombre “software” fue reemplazado por el de “firmware”, más adecuado para describir instrucciones inmodificables.

En el principio, el código debe ser creado por el programador. Se puede utilizar tanto un ensamblador como un compilador cruzado o “cross-compiler”, que se diferencia de los compiladores comunes en que producen código para una arquitectura diferente a la que utilizan para funcionar. Por ejemplo, se acostumbra que los compiladores de C que corren en ambientes PowerPC, produzcan aplicaciones que corran sólo en esa plataforma, y ocurre de igual manera con los compiladores para arquitecturas x86. Pero un cross-compiler produce código para otras plataformas, como lo hace el compilador GCC (Conocido compilador gratuito) que sobre un CPU Pentium, puede generar aplicaciones que corran en arquitecturas x86, powerPC, sparc, y muchas otras. Esto permite utilizar estos compiladores para generar código que corre en procesadores de 8 bits, que son dispositivos que no podrían realizar la compilación por si mismos. En el Capítulo 3 se detalla el compilador seleccionado.

Una vez obtenida la aplicación en forma de código binario, debe encontrarse alguna manera de hacerlo llegar al CPU que se halla dentro del microcontrolador, como se ve en la figura 2.4. Esto es sencillo en sistemas potentes como un PC, en los que la aplicación se almacena en un CD-ROM o en el disco rígido. El microcontrolador dispone de hardware especializado para adquirir su programa de una fuente externa, como puede ser una PC.

Los métodos de programación se resumen como:

- ICSP: (In Circuit Serial Programming) o programación serie en-circuito, es la opción más general.
- Bootloader: Método novedoso en el mundo de los microcontoladores, funciona de manera similar al mecanismo de booteo en una computadora normal.

2.4.2. Programador ICSP

Mediante esta característica del PIC-18, que comparte con muchos otros productos de Microchip, se transmiten los datos de programación al dispositivo por medio de un protocolo serie sincrónico. Se utiliza una línea de datos, otra línea que transporta la señal de reloj y tres líneas de control. Los voltajes de programación varían entre los 12 y 17 Volts, que son necesarios para inicializar las celdas de memoria flash, aunque este modelo dispone de una versión denominada LVP (Low Voltage Programming) que permite utilizar voltajes de programación tan bajos como 2 Volts.

Aprovechando la característica LVP, surgieron varias aplicaciones freeware que simulaban el protocolo de programación ICSP mediante el puerto LPT (Puerto de impresora) estándar, entre ellas IC-PROG, que fue la utilizada durante los primeros días del proyecto. Sin embargo se necesita cierto hardware de adaptación entre el puerto LPT y el microcontrolador, como se ve en la figura 2.5.

Es importante realizar un paso adicional antes de la programación, que es el borrado. Este proceso requiere un mínimo de 4.5 Volts aplicados al microcontrolador, pero la mayoría de los puertos LPT no alcanzan este voltaje, y el utilizado en el proyecto no fue la excepción, ya que sólo alcanzó los 3.5 Volts, que fueron insuficientes para el borrado por lo que hubo que aplicar algunas conexiones adicionales a fuentes de mayor voltaje.

Microchip vende un programador compatible con todos sus dispositivos, relativamente accesible a unos 150 U$S (Sin gastos de envío ni impuestos). Pero para reducir los

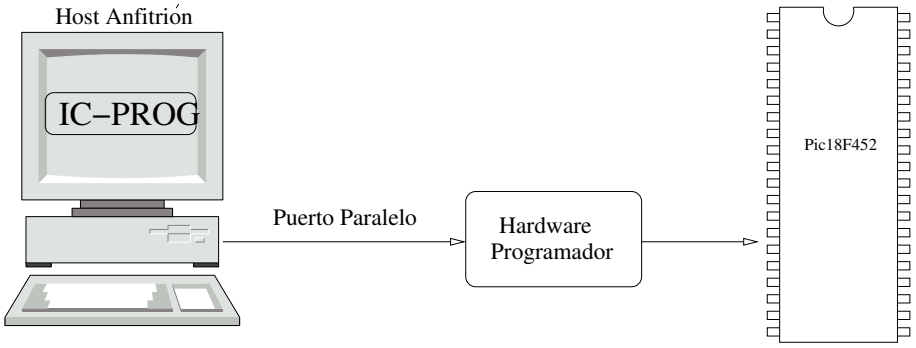


Figura 2.5.: Esquema de un programador común.

gastos del proyecto se decidió utilizar el programador gratuito citado anteriormente, IC-PROG. Es justo mencionar que la empresa ha liberado toda la información con respecto a los protocolos y métodos de programación, posibilitando la creación de este tipo de software.

IC-PROG

Este software está especializado en la programación de dispositivos flash, y cuenta con soporte para más de 200 modelos de diferentes empresas. Aunque el soporte para PIC-18 era experimental en el momento de la descarga, funcionó perfectamente seleccionando el modelo de programador 'Conquest' [9] e implementando un hardware de programación muy simple, que sólo cuenta con un par de resistencias de 'pull-down' conectadas a las líneas de datos y control. El software corre sólo en Windows 98 debido a que necesita acceso directo al hardware del puerto paralelo, aunque existe un driver también experimental, que permite su uso en Windows 2000/XP. En la figura E.7 en la página 107 puede verse una captura de pantalla del software IC-PROG.

La operación de borrado-programación-verificado de la memoria Flash completa (32 Kbytes) toma unos 3 minutos para finalizar completamente utilizando IC-PROG, tiempo durante el cual la PC debe dedicarse exclusivamente a la tarea de programación o se corre el riesgo de introducir errores debido a "time-outs". Sin embargo, se cuenta con un método adicional para acelerar este proceso.

2.4.3. BootLoader

La línea PIC-18 y los modelos más recientes de la línea PIC-16 cuentan con otro método denominado "auto-programación", que permite que el microcontrolador modifique su propio código aún mientras está en funcionamiento. Este proceso, que existió desde siempre en arquitecturas Von-Neumann, es relativamente reciente en el área de los microcontroladores. Utilizando ésta técnica surgen los llamados "BootLoaders", sistemas que

[9] Al ser un software tan flexible, hay que especificar tanto el dispositivo en cuestión a programar (Modelo, tipo, etc.) como el hardware de programación, ya que IC-PROG también soporta la utilización de distintos programadores.

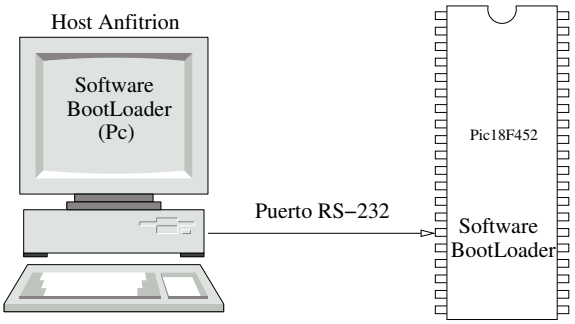


Figura 2.6.: Esquema de programación mediante un Bootloader

se cargan al inicio de la ejecución y permiten descargar software nuevo al microcontrolador utilizando los periféricos disponibles y/o ejecutar el código ya existente. Las ventajas de utilizar este software son muchas, entre ellas:

- Un “BootLoader” tiene a su disposición todos los periféricos del microcontrolador para descargar los datos, por lo que puede realizar la descarga, por ejemplo, por medio de la interfaz RS-232, acelerando en gran medida el proceso de programación.
- El software de programación es generalmente más sencillo de utilizar, ya que separa el proceso en dos módulos: uno residente dentro del microcontrolador, y otro residente en el sistema anfitrión (PC).
- Toda la línea PIC-18 tiene soporte de hardware para este tipo de software, ya que permiten proteger los primeros 512 bytes de memoria, donde se supone que se ubican los “BootLoaders”.

Este tipo de software se utiliza desde hace mucho tiempo en entornos industriales, y específicamente es usado cuando se “Flashean” (término utilizado para denominar la actualización del firmware) las memorias de muchos de los componentes de las PCs modernas.

Luego de algunos meses de utilizar el sistema ICSP mediante el software IC-PROG, se decidió por utilizar un bootloader y se escogió el denominado *Jolt Booloader*, que surgió como una modificación de la nota de aplicación[10] AN851 de Microchip (A FLASH Bootloader for PIC16 and PIC18 Devices).

Jolt BootLoader permite descargar y configurar todas las opciones del microcontrolador PIC-18 a través de la interfaz RS-232 del mismo, como puede apreciarse en la figura 2.6. La velocidad varía de los 9600 a los 115200 BPS, permitiendo la programación y verificación completa en unos 50 segundos.

Curiosamente el módulo del anfitrión (que corre en la PC) de este software está realizado completamente en lenguaje Java, que se suele considerar como un lenguaje de muy alto nivel, pero en este caso se utiliza para descargar el firmware al microcontrolador utilizando el *Java Communications API*, además del JRE 1.4 y, como todas las aplicaciones de Java, es completamente portable.

[10] Las Notas de Aplicación equivalen a los Artículos del ambiente académico, pero utilizados en el entorno privado o comercial.

2.5. Interfaz RS232

La interfaz conocida actualmente como RS-232 se estableció originalmente en 1962 por la EIA (Electronic Industries Association). Actualmente está en su sexta versión EIA-232-F de 1997. Especifica parámetros mecánicos, eléctricos y funcionales de una interfaz de comunicaciones entre dos dispositivos finales. Se denominan comúnmente puertos serie y hace unos años se utilizaban para conectar periféricos que requerían una baja tasa de transferencia de datos, tales como módems y ratones. Hoy en día todo computador posee al menos una y es uno de lo métodos más utilizados para la configuración inicial de routers y firewalls comerciales. El dispositivo de firewall diseñado utiliza la interfaz RS-232 no solamente para la configuración, sino también para la programación, como se explicó en la sección 2.4.3.

2.5.1. Especificación

Consta en su implementación más simple de 3 conexiones, una para transmitir, otra para recibir y la tercera como conexión de tierra. Los bits de datos se transmiten consecutivamente a intervalos predeterminados de tiempo, y al no poseer líneas de sincronización, deben existir señales de inicio y parada de la comunicación. Los voltajes que maneja son, de -12 a -5 Voltios para simbolizar un ’1’ y de 5 a 12 para simbolizar un ’0’. El control de flujo puede realizarse por software o por hardware (utilizando líneas adicionales) , o puede no existir, que es la opción que se escogió para el sistema.

La transmisión se denomina “Asincrónica”[11] ya que no existen señales de sincronía para indicar el inicio y finalización de los datos, y además dicha transmisión puede iniciarse en cualquier momento dado. Existen versiones sincrónicas de RS232 pero han quedado en desuso. La velocidad de esta interfaz está acotada entre los 2400 y 115200 BPS, y la distancia entre nodos no puede superar los 3 Metros sin introducir errores por interferencia ambiental.

2.5.2. Implementación

Desde el lado de la PC, están normalmente instaladas una o más de estas Interfaces mediante el conocido integrado 16550 o algún otro compatible con el mismo. Del lado del microcontrolador existen dos opciones:

- Implementar una interfaz RS-232 por software: utilizando dos de las entradas / salidas digitales de propósito general, (se puede simular una interfaz de este tipo mediante la programación adecuada). Es la única opción en microcontroladores antiguos o de gama baja. El proyecto citado en la página 19 utiliza este tipo de implementación.

- Interfaz RS-232 por hardware: Los microcontroladores modernos y de gama alta, tal como el utilizado para el proyecto actual, posee hardware dedicado a la transmisión serial. En este caso, el hardware puede configurarse tanto para actuar como RS-232, RS-422 o RS-485, ya que soporta transmisiones sincrónicas y asincrónicas.

[11] Existen estándares RS-232 tanto sincrónicos como asincrónicos.

Pin	Descripción	Pin	Descripción
1	Reset	20	+5v
2	D6	19	D7
3	D4	18	D5
4	D2	17	D3
5	D0	16	D1
6	/IOW	15	AEN
7	A3	14	/IOR
8	A1	13	A2
9	INTR	12	A0
10	Gnd	11	nc

Cuadro 2.3.: Disposición de las conexiones del bus

Aún así, la salida de la interfaz hardware no se puede conectar directamente a los cables de I/O, ya que necesita pasar previamente por una conversión de niveles: Los niveles de este Microcontrolador son CMOS, o sea, 0 volts para un '0' y el voltaje de alimentación de este microcontrolador (5 Voltios generalmente) para representar un '1'.

2.5.3. Conversor de niveles

A continuación se hará una breve reseña resultado de la investigación sobre estos componentes, con suficiente detalle para servir de guía al lector interesado.

Existen varios tipos de conversores de niveles, entre los que se destacan los conversores "económicos" formados por un par de transistores, que funcionan relativamente bien para bajas velocidades. En el proyecto anterior se utilizó uno de este tipo, debido a que se necesitaba una velocidad relativamente baja (19200 BPS). El proyecto necesita velocidades mayores[12] que necesitan de un hardware adicional, denominado *"Transciever"*, o conversor de niveles. Este tipo de hardware dispone de una red de capacitores y un circuito especial para obtener los altos voltajes, tanto positivos como negativos, que necesita la Interfaz RS-232 para funcionar adecuadamente. El integrado seleccionado fue el "HIN-232", reemplazo del original "MAX-232". Generalmente es difícil adquirir los modelos originales de circuitos integrados en la zona, y sólo se consiguen reemplazos que son de menor precio y poseen características similares, aunque generalmente inferiores, que ofrecen convertidores suficientes para un par de interfaces RS-232. En el circuito, (ver apéndice D) puede verse que sólo se utiliza una.

[12] Para la operación, bastan y sobran los 9600 BPS que utilizan, por ejemplo, todos los equipos Cisco, pero para la programación es deseable obtener la máxima velocidad, unos 115200 BPS.

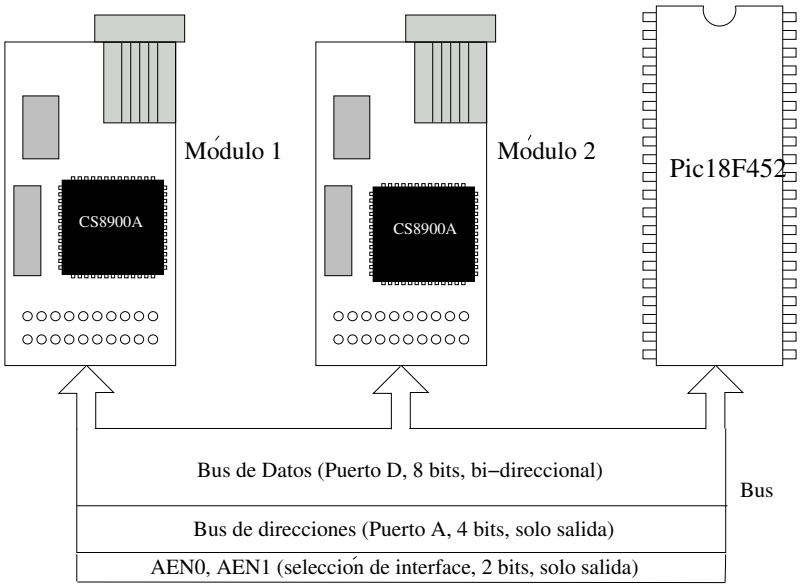


Figura 2.7.: Detalle del bus del sistema

2.6. Bus

El bus utilizado es una adaptación del Bus ISA de 8 bits. Se utiliza un conector de 20 pines en cada Interfaz y en la placa del microcontrolador, cuyo detalle se aprecia en el cuadro 2.3 y la descripción detallada es la siguiente:

(A0-A3): 4 bits de dirección

(D0-D7): 8 bits de datos

IOW, IOR: indican una lectura o escritura sobre el Bus

AEN: En la especificación figura como “Address Enable” y se utiliza cuando el subsistema DMA toma el control del bus. Como el circuito no dispone de tales mecanismos, es utilizado como un “chip select” para activar o desactivar la Interfaz.

INTR: (Interruption Request): señaliza la producción de una interrupción por hardware. Según la nota de aplicación AN-181 de Cirrus Logic, el CS8900A no debe utilizar interrupciones durante su funcionamiento en modo de 8 bits, ya que debido a un error de hardware se producirían errores en la transferencia. En este modo se deben usar exclusivamente técnicas de pooling, por lo que en el circuito no se utiliza esta conexión.

Gnd, +5v: La alimentación de las interfaces se realiza por medio de estas dos líneas. Se debió agregar capacitores de filtro entre estas dos líneas en sendas interfaces, ya que de lo contrario se producía demasiado ruido y el microcontrolador presentaba un comportamiento errático.

Nc: No conectar (Pin libre).

Listado 2.1: Algoritmo de la transferencia en bloques

```
void movepacket(void)
{
while(queden bytes para transmitir)  {
    Seleccionar_interfaz_fuente();
    leer_bloque(); //Actualmente se leen bloques de 128 bytes
    Seleccionar_interfaz_destino();
    escribir_bloque();
    }
}
```

En la figura 2.7 se detalla la disposición del bus entre los distintos componentes del sistema.

2.6.1. Transferencia entre microcontrolador y Ethernet

El mecanismo por el cual se decide o no la transferencia de datos se relata con detalle en el capítulo 9 en la página 84, sin embargo, se dará una idea general de la transferencia de alta velocidad que se realiza entre las Interfaces, como preámbulo a las optimizaciones que se pueden realizar. El cuello de botella de la transferencia de datos se encuentra en el algoritmo (Listado 2.1) que se utiliza cuando ya se decidió dejar pasar un paquete. Puede apreciarse que el método actual utiliza intensivamente el procesador para mover la información de un módulo al otro, transfiriendo bloques de 128 bytes a través de la memoria.

Dicho comportamiento se puede ver en la figura 2.8 a), que ejemplifica el movimiento de datos realizado con el algoritmo. Puede verse que la transferencia es “half-duplex”[13], o sea, sólo una de las interfaces está activa a la vez, de la cual se lee o se escriben los datos del paquete.

Utilizando el bus en su configuración actual, se obtiene una transferencia aproximada de 2 megabits por segundo en promedio, velocidad más que suficiente para cumplir con los objetivos planteados.

[13]No confundir con la configuración de las Interfaces, que siempre serán “full-duplex”

2.7. Optimizaciones posibles

Una manera de acelerar la transferencia es la siguiente: en lugar de activar una interfaz a la vez, se podrían activar ambas, y realizar la transferencia sin que los datos deban pasar por el microcontrolador, imitando el comportamiento del DMA (Direct Memory Access, acceso directo a memoria) en sistemas mayores. De esta manera se ahorraría el tiempo de la copia de los datos, y el microcontrolador sólo actuaría como arbitro del bus, manejando las líneas de control y dirección, como se observa en la figura 2.8 punto b). Esto puede acelerar en gran medida las transferencias, ya que, como se dijo anteriormente, es el cuello de botella del todo el sistema de firewall. Para acotar el proyecto, queda pendiente de realización futura.

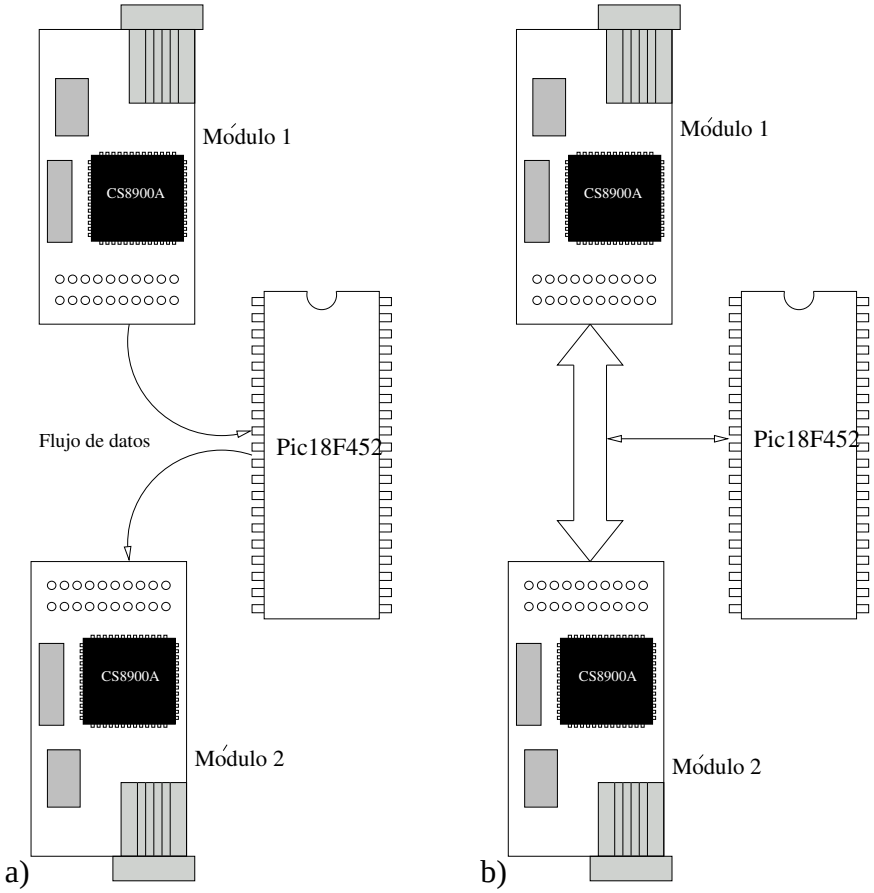


Figura 2.8.: a)Transferencia actual b)Posible optimización

3. Compilador y Sistema Operativo

En éste capítulo se describirá en detalle el diseño y selección del sistema operativo, tarea en este caso muy relacionada con la selección del compilador y lenguaje utilizado, tanto que no pueden tratarse por separado.

3.1. Lenguaje de Programación y Compilador

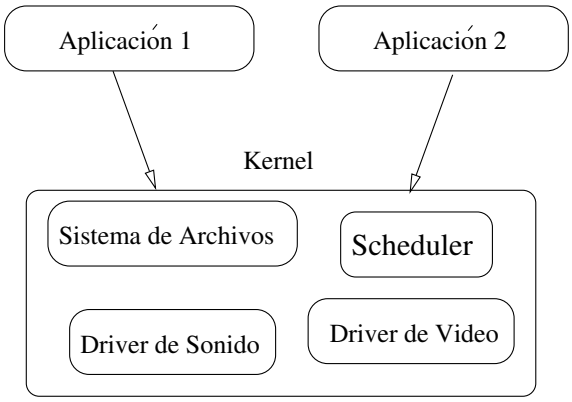
Una vez finalizada y probada la construcción del hardware, se comenzaron las pruebas preliminares de código sobre el microcontrolador, durante las cuales se intentó seleccionar la plataforma de desarrollo más adecuada.

3.1.1. Lenguaje a utilizar

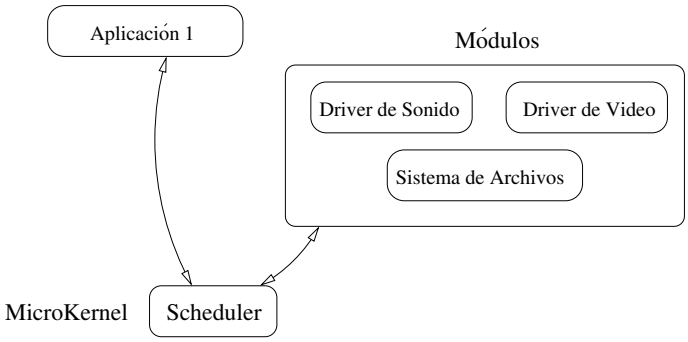
Hasta poco tiempo atrás, el lenguaje excluyente para trabajar con microcontroladores de gama baja era el código ensamblador, debido a los pocos recursos con que se contaban. Existían compiladores de C, pero no cumplían con el estándar ANSI y generalmente producían código de menor calidad, aunque, por supuesto el código fuente era mucho más claro y mantenible. Debido a la popularidad de la familia PIC de Microchip surgieron muchos otros lenguajes, un poco experimentales, entre los que se cuentan PASCAL, BASIC y algunos no-estándar tal como JASP.

La familia PIC18 de microcontroladores posee características avanzadas, entre las que se destaca la capacidad de manejo de su pila de hardware, y registros especialmente diseñados para su utilización por un compilador de alto nivel, que utiliza una pila de software, tal como C. De hecho es la primera característica que se nombra en el manual del microcontrolador (en la página 3), ya que el fabricante le asigna mucha importancia a este hecho presentándolo como “optimizado para la programación en C”. Desde el principio se seleccionó este lenguaje para la programación.

Existen hoy en día varios compiladores para el microcontrolador PIC-18, disponibles como versiones de demostración, pero completamente funcionales, entre los que se destacan HI-TECH PICC18, y Microchip C18. Aunque con precios y prestaciones aparentemente similares ,existen diferencias fundamentales entre los dos compiladores que se detallan más adelante, por ahora se dirá que el seleccionado fue Microchip C18, ya que es el único que permite la utilización de un sistema operativo completamente preemptivo.



(a) Kernel monolitico



(b) Micro-Kernel

Figura 3.1.: Diferentes tipos de sistema operativo

3.1.2. Idioma

Como convención se adoptó el idioma ingles para nombrar las variables, funciones y realizar los comentarios del código fuente, con el objetivo de lograr una mayor difusión, aunque en la interfaz al usuario se utilizó el lenguaje Castellano, para facilitar las demostraciones locales.

3.2. Elección de sistema operativo

Durante el principio del desarrollo de la tesina, se dedicó especial atención a la elección del software de base, ya de esta decisión dependería en gran medida el desarrollo del resto del proyecto. Se dará a continuación una breve reseña de los tipos de sistemas sobre los que se tuvo que decidir:

Sistema Operativo: Una aplicación que utilice este tipo de software se organiza como un sistema de capas, en la que cada una provee servicios a la capa superior. Existen muchos tipos, entre ellos:

- Cooperativo: En este tipo de Sistemas Operativos los procesos cooperan entre sí para repartirse los recursos del sistema. Una desventaja es que un proceso 'egoísta' que se apropia de algún recurso como lo es el CPU no puede ser interrumpido, y su mal funcionamiento afecta a todo el sistema. Como ejemplo, existe Windows 3.1, de Microsoft.
- Preemptivo: En este tipo el sistema asigna y quita recursos a los procesos, sin que ellos puedan evitarlo. Para realizar esta tarea se necesita soporte de hardware, que debe proveer de algún medio para quitar el control a un proceso, como puede ser una interrupción de hardware. Windows 95 es un ejemplo típico de estos de sistemas.
- Monolítico: Considerando una diferente clasificación ilustrada en la figura 3.1, en la que se tiene en cuenta las capas y la separación entre ellas. En sistemas monolíticos, el conjunto de servicios de un sistema operativo se concentra en un solo espacio de direcciones, en el que existen funciones que las aplicaciones deben llamar. En este espacio conviven todos los drivers, junto con los servicios tales como sistemas de archivos, etc. (ver Tanenbaum:1992). Un ejemplo puede ser el sistema operativo Linux (aunque en versiones recientes adopta algunas características modulares de los sistemas microkernels).
- MicroKernel: Siguiendo la misma clasificación que la opción superior, en este caso el núcleo o kernel está reducido al mínimo posible, y la mayoría de los servicios (Drivers, etc.) se implementan como módulos en un espacio de direcciones distinto, por lo que un mal funcionamiento de un Driver queda aislado y no afecta el resto del sistema (ver Tanenbaum:1992). Windows 2000/XP está basado en un sistema de este tipo.

Superloop: Este sistema es simplemente un método de simular la ejecución concurrente de varios procesos, ejecutándolos consecutivamente dentro de un bucle. Podría verse como la versión elemental de un sistema operativo cooperativo.

3.2.1. Sistema operativo Vs. Superloop

El autor está familiarizado con la programación de aplicaciones sin utilizar los servicios de un sistema operativo, al haber realizado numerosos prácticos en lenguaje ensamblador sobre DOS. Este sistema permite que una aplicación acceda directamente al hardware. Sin embargo, las aplicaciones de red son inherentemente concurrentes, por lo que es necesario un soporte multi-tarea o multi-hilo para realizar una implementación elegante.

3.2.2. Superloop

Según la bibliografía y experiencia propia, las aplicaciones para microcontroladores siempre siguieron el patrón de diseño de Superloop, en el cual cada tarea se realiza secuencialmente dentro de un loop infinito maestro. Utilizando este método no es posible utilizar operaciones bloqueantes, por lo que el software resultante es diferente al que se realizaría sobre un SO real, por ejemplo, sobre Unix.

Listado 3.1: Superloop tipico

```
void main()   {
  while(true) {
     if (kbhit()) getch();
     if (pktReceived()) processPacket();
     if (temp>30) startCooler();
     }
  }
```

Listado 3.2: Código utilizando un Sistema Operativo

```
void main()
  {
  startTask(readKeyboard());
  startTask(readPackets());
  startTask(controlTemperature());
  while(true) continue;
  }
```

En el ejemplo que puede verse en el Listado 3.1, las rutinas de lectura de teclado, procesamiento de paquetes y un sensor de temperatura se llaman desde un bucle, con la misma prioridad.

3.2.3. Sistema Operativo

Con un sistema multi-hilo, el código podría escribirse de manera muy diferente, de manera que cada tarea quede aislada, con todas las ventajas que tiene la “encapsulación”, como se ve en el listado 3.2.

Existen varias ventajas al utilizar un sistema operativo, aparte de la ya mencionada “encapsulación” de los procesos, tales como la disponibilidad de servicios de sincronización, y un mejor aprovechamiento general de los recursos (CPU, periféricos, etc) ya que son administrados por una aplicación específica.

Se tomó la decisión para el proyecto, de utilizar un sistema operativo y se comenzó una investigación para averiguar las opciones disponibles. Dicha investigación fue realizada exclusivamente sobre Internet en el período diciembre del 2002-febrero del 2003[1].

Una rápida búsqueda eliminó la mayoría de los sistemas operativos de tiempo real, tal como OS-9 y QNX, ya que algunos fueron diseñados para procesadores de 32 bits, al igual que todas las versiones de Linux para microcontroladores[2] y son escasos los que soportan el dispositivo PIC18F452. Sin embargo, este modelo permite en teoría todos los tipos de sistemas nombrados en la sección 3.2, con la excepción del sistema de MicroKernel, ya que no dispone de los mecanismos de hardware necesarios para un aislamiento completo del núcleo y los módulos.

3.2.4. SALVO (Pumpkin Inc.)

Mediante esta búsqueda se halló un sistema operativo denominado SALVO, que funciona en todos los modelos de Microchip (Inclusive los de gama baja, tal como el PIC16F84), y está disponible una versión gratuita para su descarga. Según el manual de dicho sistema operativo, está diseñado para su utilización con el lenguaje C, y soporta todos los compiladores disponibles para PICs.

SALVO resultó ser un software muy interesante: Es un sistema operativo cooperativo, con muchos servicios tales como semáforos, delays y colas de mensajes. Ocupa muy poco código (de 500 instrucciones hasta 3000, depende de las opciones compiladas) y sólo algunas decenas de bytes de memoria. Por supuesto tiene sus desventajas: para poder utilizar a SALVO en sistemas que ni siquiera tienen pilas de hardware para las llamadas, se utiliza un sistema de saltos incondicionales hacia su scheduler, y no soporta el cambio de contexto[3] desde ninguna función que no sea la principal del proceso, en otras palabras, no soporta la utilización de sub-rutinas. Esto, unido a las limitaciones de la versión gratuita (sólo 3 procesos, etc.) llevaron a la búsqueda de otro sistema, aunque en la primera fase del desarrollo, se utilizó a Salvo como kernel para la realización de otros sub-sistemas, tales como el sistema de archivos y el shell.

3.2.5. uC OS - II y otros

Para ésa época, existía otro sistema operativo, llamado uC-OS, pero no estaba disponible gratuitamente, sino a través de la compra de un libro, -operación bastante complicada desde Argentina, la región donde fue desarrollado el proyecto-. uC-OS es un sistema que funciona en multitud de microcontroladores, entre ellos la familia PIC18 de Microchip, y es completamente preemptivo. Para febrero del 2003, uC-OS estaba disponible para su descarga gratuitamente, pero no fue utilizado debido a que ya se había desarrollado un sistema propio.

[1] Es importante esta referencia temporal, ya que al ser un dispositivo relativamente nuevo en el mercado, la disponibilidad de software para el MCU PIC18F452 aumentó apreciablemente durante la realización de la Tesina.

[2] Además, todos estos sistemas necesitan, en promedio, un mínimo de 4 Mbytes de ROM (o Flash) y 2 MB de RAM.

[3] Como el lector podrá adivinar, SALVO en realidad no realiza ningún cambio de contexto, sino un simple “goto” al siguiente proceso. Por eso todas las variables deben ser globales al utilizar este software.

Actualmente existen multitud de sistemas operativos para el Microcontrolador PIC-18, entre ellos Embeddinet PRO[4], Real-Time Architect para PIC18 y PICOS18, éste último bajo licencia GPL.

3.2.6. Decisión adoptada

Como se dijo en la sección 3.2.2, primeramente se utilizó al sistema Salvo como base para el desarrollo de software, pero a medida que se avanzaba en el desarrollo las limitaciones se hicieron evidentes y al no existir otra opción, se comenzó a considerar la implementación de un sistema operativo propio. Esta tarea fue realizada paralelamente al desarrollo con el sistema Salvo, para tener una opción disponible en el caso de que fracasaran los intentos de creación del kernel. Sin embargo se logró una implementación funcional y fue la utilizada finalmente.

3.3. Diseño del Kernel

Se realizaron varios intentos de desarrollo e implementación de un kernel multi-hilo para el PIC-18, y, luego de reiterados fracasos se logró la realización de uno perfectamente funcional. Pudo observarse una característica de los sistemas operativos en tiempo real: La ejecución y sincronización debe ser perfecta, o de lo contrario se bloquean inmediatamente.

Las siguientes metas fueron fijadas desde un principio:

- Sistema Preemptivo Multi-Hilo
- Delays (Retrasos y llamadas bloqueantes)
- Implementación en C mientras sea posible.

Se puede ampliar este último punto: Todo el código fuente está escrito en el lenguaje C, pero contiene algunas instrucciones de assembler del tipo 'in-line' o sea, embebidas dentro del archivo .C porque implementar algunas operaciones (tales como el 'POP' de assembler) es imposible utilizando el lenguaje C puro.

3.3.1. MMU

Al no poseer el PIC-18 una MMU (Memory Managament Unit, ver Tanenbaum:1992) no es posible el aislamiento total de procesos, al estilo Unix o Windows 2000[5]. Sin embargo, se puede lograr un sistema multi-hilo, o sea, desde el punto de vista del programador, el PIC-18 es un proceso que puede crear varios hilos de ejecución, y cada uno puede acceder y modificar cualquier variable global del PIC, como un hilo del sistema Linux, que puede acceder a las variables globales del proceso al que pertenecen.

[4] Este software, que ofrece funcionalidades similares al desarrollado, tiene un precio de licencia mayor a los u$s 700

[5] Sin embargo, a partir de la compilación (Build) 600 del sistema, la aislación lograda por el Scheduler fue lo bastante buena como para que no se registrara ninguna pérdida de control total, sino cuelgues aislados en los procesos.

3.3.2. Tiempo Real

Como se dijo en la sección 1.4.2 en la página 17, para que un sistema operativo sea considerado de tiempo real, su kernel debe ofrecer una latencia constante, no mínima, sino constante, para que se pueda calcular, "determinísticamente" cuándo una tarea se ejecutará. Quizás una aplicación de alto riesgo como la operación de una central nuclear requiera de esta precisión, pero un Firewall no tiene tantas exigencias temporales, por lo que no se consideró esta restricción.

Esencialmente, el kernel sólo provee dos servicios: Retardos (Delays) y el Scheduler, que no es realmente un Servicio, pero simula la ejecución concurrente de varios hilos. Los demás módulos, como el Sistema de Archivos y la pila TCP/IP están muy relacionados entre sí, aunque podrían funcionar por separado utilizando otro Kernel, con algunas adaptaciones mínimas.

3.3.3. Scheduler

Esta parte esencial del kernel, debe realizar el característico "cambio de contexto", o sea, guardar toda la información concerniente al proceso que se estaba ejecutando, y cargar la información del nuevo proceso a ejecutar. Puede intuirse que se requieren grandes cantidades de memoria para almacenar la información de un proceso. Esto es cierto y es uno de las desventajas de éste tipo de sistemas multi-hilo, pero en general no es necesario guardar absolutamente todo el contexto de un proceso, característica que se utilizará para ahorrar memoria.

Se repasará el diseño original del Scheduler, que no difiere mucho del diseño final: Es una función monolítica ISR (Interruption Service Request, Pedido de servicio de Interrupción) o sea, corre como resultado de una Interrupción, que en éste caso es activada por un Timer periódicamente. El pseudocódigo puede verse en el listado 3.3.

Todo el sistema fue compilado mediante HI-TECH PICC18, y no funcionó de ninguna manera. Aunque se refinara el código, se guardaran multitud de registros y se controlara concienzudamente que se restauraran la pila y los registros perfectamente, los procesos "morían" irremediablemente. El problema, para alguien sin experiencia en el diseño de sistemas operativos, era prácticamente invisible, hasta algún tiempo después cuando se profundizó en el estudio y la comprensión del funcionamiento de los compiladores.

3.3.4. El Problema

Básicamente, el problema estaba en la definición de lo que es el contexto para un PIC-18:

Registros: de los cuales como se dijo, no hace falta guardar todos sino unos 10 de los aproximadamente 90 que posee.

Pila de Hardware: Almacena las direcciones de retorno de hasta 32 llamadas anidadas. Existe una pila que posee un solo nivel, llamada Pila de Contexto Rápido, cuya función es almacenar algunos registros y una dirección de retorno, para ser utilizada en interrupciones de baja latencia, ya que guarda y restaura todos los datos en una sola oscilación del reloj. Pero se verificó que esta pila no fuera utilizada en ningún momento en el programa, así que ése tampoco era el problema.

Listado 3.3: Pseudo-Código del scheduler

```
void ISR(void)

{

Desactivar Interrupciones

Guardar Contexto:

Guardar Registros

Guardar Pila de Hardware

Seleccionar siguiente Proceso (Al principio, Round Robin)

Cargar nuevo contexto:

Cargar Registros

Cargar Pila de Hardware

Activar Interrupciones

}
```

Variables globales: Son asignadas por el Compilador a direcciones fijas de memoria, y no necesitan ser salvadas, ya que en realidad, no pertenecen al contexto de un hilo, sino que son compartidas por todos.

Variables locales: Este era el problema. Se había omitido el guardar la *pila de software*, que es donde se guardan los parámetros, las variables locales (llamadas 'automáticas' en C, ver Sebesta:1996) y los valores de retorno de las funciones. Éste es un tema estudiado a fondo en la asignatura 'Lenguajes de Programación' y su comportamiento en los compiladores es prácticamente estándar y bien comprendido.

Ya identificado el problema, solamente quedaba ubicar el lugar en la memoria donde el compilador ubica su pila de software, y salvarla en alguna otra dirección para que sea reemplazada por la pila de software del nuevo proceso. Pero causó sorpresa cuando, luego de una ardua investigación sobre la documentación del compilador, se descubrió que el compilador PICC18 de HI-TECH no utiliza ningún tipo de pila de software, sino una tecnología muy diferente, que se explicará a continuación.

3.3.5. La pila compilada de HI-TECH

A lo largo de muchos sitios Web, se asegura que los compiladores de HI-TECH son los que producen el código más compacto del mercado, y por este motivo su elevado precio,

de más de 1000 U$S[6]. Una razón muy importante para ello (además de librerías muy optimizadas) es la utilización de una P*ila Compilada y un Árbol de llamadas* (Call-Graph) para realizar todos los pasajes de parámetros y ubicación de variables automáticas. (Esta metodología no es utilizada en los compiladores para sistemas mayores, que disponen de memoria relativamente ilimitada).

El funcionamiento es el siguiente:

En el momento de la compilación, el compilador analiza el código y realiza un árbol de llamadas, con la función `main()` en la raíz del árbol, y todas las funciones tal como son llamadas. De esto se desprende que éste sistema no permite ningún tipo de recursión. Entonces, al conocer el orden en que las funciones son llamadas, es posible asignar estáticamente posiciones de memoria a todas los parámetros, variables automáticas y valores de retorno de las funciones, teniendo cuidado de que nunca se solapen. Como resultado, todas las variables automáticas son en realidad, estáticas, aunque sólo son válidas durante la ejecución de la función, ya que pueden ser reutilizadas por otra función que se ejecute en otra rama del árbol.

Este modelo tiene las siguientes ventajas con respecto al uso de una pila de software normal:

- Se reduce el código necesario, ya que no hace falta manejar una estructura adicional como es la pila de Software.
- Se aumenta la velocidad de ejecución, por la misma causa anterior.

Y las siguientes desventajas:

- No se permite ningún tipo de recursión, ya que esto violaría el árbol de llamadas.
- Las funciones de alto orden (Punteros a funciones) son de difícil manejo y complican de manera enorme el árbol de llamadas, así como las funciones llamadas desde ISRs (Aunque el compilador de HI-TECH lo maneja muy bien).
- No se permiten los sistemas operativos preemptivos, ya que esto supone que se rompa el árbol de llamadas para pasar a ejecutar otra función, lo que dejaría todas las variables y parámetros en un estado inconsistente. SALVO (ver sección 3.2.2), que es soportado por HI-TECH, recomienda sólo utilizar variables estáticas en los procesos.

3.3.6. Solución

El problema era serio, no solamente no se guardaba el contexto, sino que era imposible hacerlo con ese tipo de compilador.

Afortunadamente, el otro compilador disponible creado por el fabricante, Microchip C18, es un conocido compilador con pila de software, completamente apto para su uso con un sistema operativo, por lo que a partir de ése momento (mediados de enero de 2003) se utilizó sólo éste compilador, que como todo software, también tiene algunos problemas:

[6] Como todo en el mundo de la informática, este precio está bajando de manera constante.

1. El código C no es compatible entre los dos compiladores, no sólo por problemas de nombres en las librerías, sino que Microchip hace diferencias entre punteros a RAM y a ROM, y HI-TECH tiene punteros genéricos.

2. El uso de una pila de software provoca código adicional y no tan optimizado como el de HI-TECH. Ej.: La longitud del una porción del código del sistema era de 11 Kb compilado con HI-TECH, y la misma pasó a cerca de 18 Kb compilado con C18 de Microchip

3. Las opciones de debug de Microchip son inferiores a las del entorno de HI-TECH, específicamente no se simula la interfaz serie, por lo que el desarrollo del software se vió muy perjudicado, ya que no es posible simular la mayoría del comportamiento y debe ser probado directamente sobre el microcontrolador real.

Algunas desventajas fueron subsanadas con versiones posteriores de C18, por ejemplo, el código Paso de 18 Kb con la versión 2.00 a 11 Kb con la versión 2.10.2, debido a multitud de nuevas optimizaciones.

> *NOTA:* Microchip tiene disponible una versión de prueba de su compilador C18, válida por 30 días sin ninguna restricción adicional. El IDE (Integrated Development Enviroment, Entorno Integrado de Desarrollo) llamado MPLAB es de libre distribución.

3.3.7. Scheduler final

Se agregó al scheduler inicial, el código para salvar la pila de Software. Inicialmente se agregó código para mover toda la porción de memoria correspondiente a la pila, lo cual elevaba enormemente el tiempo de cambio de contexto, hasta casi 5000 instrucciones[7] para una pila moderada de 128 bytes, pero más tarde se optó por modificar sólo el puntero de la pila a la nueva posición de memoria, operación mucho más rápida y eficiente que llevar toda la memoria a la posición del puntero.

El puntero de la pila está conformado por cuatro registros, FSR1L y FSR1H que mantienen el tope de la pila, y FSR2L y FSR2H que almacenan el tope de la función actual o *'function frame pointer'*. Con sólo guardarlos junto a los demás registros especiales, se logra un cambio de contexto en aproximadamente 1000 oscilaciones de reloj.

La versión final está realizada utilizando muchas optimizaciones en assembler, y realiza un cambio de contexto en 700 instrucciones (incluidas ciertas estadísticas que el scheduler realiza).

Surgieron muchos otros problemas hasta lograr esta versión final, como la solución de bugs de hardware[8] y el perfecto timing de las interrupciones[9] que elevaron la dificultad, además de que se agregó otra restricción adicional:

[7] Se utiliza el termino "instrucción" para abstraerse de una medida temporal, ya que el tiempo real dependerá de la velocidad del oscilador. En éste caso, el PIC18F452 ejecuta 10 MIPS (Millones de instrucciones por segundo)

[8] Específicamente, si el sistema operativo no limpia el registro TBLPTRU al iniciar un proceso, se produce una corrupción en la lectura de memoria.

[9] Ej.: se refresca primero al timer o se cambia de contexto y luego se refresca el timer? ver sección 3.3.11 en la página 47.

Listado 3.4: OS-switch, guarda el contexto actual

```
void OS_switch(void)

{

1.Desactiva las interrupciones.

Esto es necesario porque sino si se interrumpe el microcontrolador
     en el medio de un cambio de contexto puede quedar en un
    estado inconsistente. Esta operación es quizás un poco
    exagerada y baste con desactivar al Timer.

2. Guardar el contexto:

Guardar registros especiales.

Guardar punteros a la pila de Software

Guardar pila de Hardware

3.Recolectar estadísticas:

Como este es un sistema en desarrollo, se necesitaran muchos datos
     para efectuar el debugging, tales como tamaño máximo de las
    pilas, porcentaje de utilización del microcontrolador, etc.

4.Llamar a OS_sched para cargar el siguiente proceso.

}
```

- El Scheduler debe ser una función que puede ser llamada tanto por un ISR como por una función de usuario.

Con esto se logra el siguiente comportamiento:

Si el scheduler es llamado desde un ISR se tiene un sistema preemptivo, que puede cambiar de contexto en cualquier momento de la ejecución de un hilo.

Por el contrario, si es una función de usuario la que llama al scheduler, se cambia de contexto solamente en puntos determinados del código, y sólo si se lo desea, por lo que en éste caso se tiene un sistema operativo cooperativo.

Una combinación de las dos opciones resulta en la mayor flexibilidad, y es llamado sistema operativo preemptivo *Mixto*.

> *NOTA:* Este sistema operativo no es preemptivo en el sentido absoluto, ya que si se desea que una función obtenga el control total del microcontrolador, solamente se debe detener el timer. No hay manera de evitar que esto suceda ya que este modelo de microcontrolador no tienen soporte de hardware para lograr 'anillos' de protección, como sucede en la arquitectura x86 de Intel, y otras.

Listado 3.5: OS-sched, selecciona el próximo proceso a correr y carga el contexto del mismo

```
void OS_sched(void)

{

1.Selecciona el siguiente proceso a correr.

2.Restaura el contexto:

Restaura la pila de hardware

Restaura los registros especiales

Restaura los punteros a la pila de Software

3.Activa las interrupciones

}
```

Se dividió el Scheduler en dos funciones:

1. `OS_switch()`: Guarda el contexto de un proceso, como se ve en el listado 3.4
2. `OS_sched()`: Carga el contexto del nuevo proceso a ejecutar, detallado en el listado 3.5

A su vez, existen otras funciones de apoyo, que son:

1. `OS_init()`: Inicializa variables del sistema y setea el hilo principal de `main()`.
2. `selectNextProcess()`: Llamada por `OS_sched()`, selecciona el siguiente proceso a ejecutar, en base a un algoritmo simple de prioridades (En las primeras versiones, se utilizaba el conocido *round-robin*, en el que todos los procesos tenían la mísma prioridad).
3. `OS_createTask()`: Setea todas las estructuras necesarias para iniciar un proceso, detallado en el listado 3.9.
4. `OS_replace()`: Reemplaza el proceso que lo ejecuta por otro, liberando toda la memoria utilizada por el proceso saliente.
5. `OS_kill()`: Maneja el estado de un proceso, pudiendo pausarlo o reiniciarlo. La eliminación aún no se implementó.
6. `OS_renice()`: Modifica la prioridad de ejecución de un proceso.
7. `OS_rpt()`: Imprime la lista de procesos, junto con la pila de llamadas e información estadística acerca de cada uno.

Listado 3.6: OS-sched,Ejemplo

```
void main()

{

OS_init();

OS_CreateProcess(proc1,...);

OS_sched();

printf(hola);

}
```

`OS_sched()` retorna directamente al punto donde se llamó a `OS_switch()` en un cambio de contexto anterior, o al comienzo del proceso, si es que se lo llama por primera vez. Es importante notar que `OS_sched()` nunca retorna a donde se lo llamó, como puede verse en el listado 3.6.

En ese ejemplo, la ejecución pasa directamente a la función `proc1()`, el hilo de `main()` es eliminado y nunca se llega a imprimir 'hola'.

De hecho, éste era el comportamiento en versiones antiguas, pero se decidió que la función `OS_init()`, además de inicializar las estructuras del scheduler, iniciara a `main()` como el proceso número 0 (cero), por lo que `OS_sched()` no debiera utilizarse nunca desde una función de usuario.

> *Nota sobre la nomenclatura:* El hecho de que las funciones del sistema operativo comiencen con el prefijo "`OS_`" no es fortuito, ya que todos los sistemas operativos que fueron estudiados (Salvo, UC-OS, etc.) llamaban a sus funciones de esta manera, como si fuera algún tipo de estándar implícito. No se descarta que exista algún estándar en el diseño de sistemas operativos de tiempo real que fue pasado por alto durante la investigación.

3.3.8. Modo preemptivo

Para configurar al sistema operativo en modo preemptivo, se debe programar algún timer de los cuatro disponibles para que produzca una interrupción periódica no inferior a unas 4000 oscilaciones de reloj, por dos motivos:

1. De reducirse esta cifra se estaría consumiendo demasiado tiempo en realizar los numerosos cambios de contextos necesarios, por lo que se reduciría el tiempo disponible del CPU para los procesos.

2. Se corre el riesgo de que bajo ciertas configuraciones, el ISR se interrumpa a si mismo, causando comportamientos impredecibles.

La función ISR utilizada para este modo puede verse en el listado 3.7.

Listado 3.7: ISR actual

```
void ISR(void)
{
ClrWdt(); // Limpiar el WatchDog Timer (Evita bloqueos continuos)
setTimer(); // Habilita timer de interrupción
OS_switch(); // Realiza cambio de contexto.
}
```

En el mismo se aprecia el orden de ejecución: Primeramente se activa el timer y luego se cambia de contexto. Si se hiciera al revés, se corre el riesgo que `OS_switch()` no retorne a la función ISR (como en el caso de que salte directamente al comienzo del un hilo) y nunca se reactivará el timer.

3.3.9. Modo cooperativo

En este modo, no hay que activar ningún timer, sino simplemente configurar los procesos y llamar a `OS_switch()` en un loop infinito dentro de cada uno de ellos, como se hace en el listado 3.8.

3.3.10. OS_CreateTask

Para finalizar, el funcionamiento del scheduler no puede ser comprendido del todo si no se explica como se inicia un proceso. El listado 3.9 ofrece un pseudo-código de todos pasos necesarios.

De esta manera, `OS_CreateTask()` deja inicializado el proceso listo para su ejecución cuando la función `selectNextProcess()` la escoja.

En el listado 3.10 puede verse un ejemplo de la creación de un proceso, donde `SofStack` debe ser un array estático de caracteres.

Esto asignará una pila de 128 bytes al hilo '`proc1`'. Si se llegara a desbordar la pila, por ejemplo, por un exceso en la recursión o demasiadas variables automáticas, posiblemente se produzca una corrupción de la memoria, por lo que este valor debe ser seleccionado adecuadamente.

Una ayuda la puede efectuar la función `OS_rpt()`, que imprime una lista de los procesos junto con la utilización máxima de la pila y otros valores, para que pueda ajustarse y no se desaproveche la memoria RAM que es uno de los recursos más escasos en el microcontrolador.

3.3.11. Time-Slice

El servicio de scheduler del sistema operativo, divide el tiempo del procesador en "rodajas" (Time-Slices) que luego asigna a los diferentes procesos de acuerdo a sus prio-

Listado 3.8: Modo cooperativo, ejemplo

```
char cont;
void main(void)
  {
  OS_createProcess(proc1,...);
  while(true)
    {
    cont++;
    OS_switch();
    }
  }

void proc1(void)
  {
  while(true)
    {
    printf(â %i â,cont);
    OS_switch();
    }
  }
```

Listado 3.9: OSCreateTask, creación de una tarea

```
void OS_CreateTask(void (*proc)(void),unsigned char prio,unsigned
    char *SofStack)

  {

  Selecciona una entrada libre en la lista de procesos.

  Asigna la prioridad al nuevo proceso.

  Inicializa la pila de Hardware con la dirección de *proc

  Inicializa la pila de Software con la dirección de *SoftStack

  }
```

Listado 3.10: OSCreateTask, Ejemplo

```
char proc1Stack[128];

void main(void)

  {

  OS_init();

  OSCreateTask(proc1,10,proc1Stack);

  while(true) continue;

  }
```

ridades. La granularidad de dicha división debe ser calculada porque si cada “rodaja” es demasiado grande, se perderá la sensación de simultaneidad de los procesos, mientras que si es demasiado chica, se perderá mucho tiempo realizando el proceso de cambio de contexto. El sistema tiene configurado un Time-Slice de 1 milisegundo (10000 Instrucciones)[10], por lo que cada segundo es dividido en 1000 segmentos que son asignados a los procesos subsecuentemente. Sin embargo, como se ve en la figura 3.2, existen dos maneras de configurar el scheduler :

a) Asignar 1 milisegundo a la suma del cambio de contexto y el proceso de usuario, o

b) Asignar 1 milisegundo exactamente al proceso de usuario, mientras que el cambio de contexto queda fuera del cálculo.

[10]Este valor puede modificarse desde el archivo `kernel/picix.h`

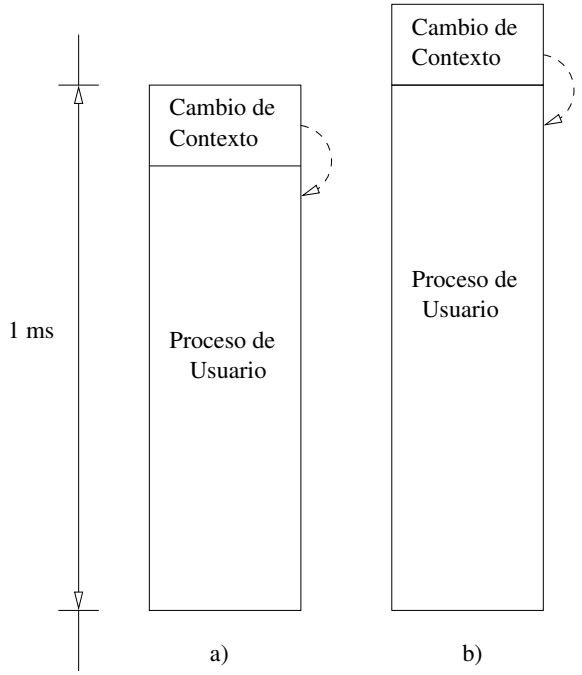


Figura 3.2.: Diferentes configuraciones de Time-Slice

Considerando que el tiempo de cambio de contexto es variable, si se utiliza la opción a) no se garantiza un tiempo constante asignado a cada proceso, mientras que si se utiliza la opción b) sí lo garantiza, pero no se pueden establecer retardos de precisión, debido a que el tiempo de cambio de contexto no es tenido en cuenta para realizar los cálculos.

La elección realizada puede deducirse del listado 3.7, que es como se estructura realmente el ISR en el sistema. El hecho de setear primeramente el timer y luego llamar al cambio de contexto, está escogiendo un time-slice correspondiente a la opción a), para de este modo poder calcular retardos con precisión.

3.4. Funcionalidad faltante

El sistema operativo desarrollado ofrece una funcionalidad mínima, aunque adecuada para el proyecto encarado, teniendo en cuenta las limitaciones de hardware. Entre las características que fueron dejadas de lado se destacan:

Comunicación entre procesos: No se implementó ningun tipo de comunicación entre procesos, ya que el sistema no la necesita a excepción de algunos casos aislados, donde se comparte una región de memoria y se protege mediante semáforos tipo "*mutex*", que no son administrados por el kernel (Ej. Comunicación entre el proceso correspondiente a la pila IP y el proceso servidor web).

Manejador de memoria: El "Memory manager" o manejador de memoria esta totalmente ausente, por lo que funciones estándar de C como `malloc()` y `free()` no se encuentran implementadas. De todas maneras no existe memoria disponible suficiente como para justificar la utilización de un servicio de este tipo.

3.5. Ubicación de archivos

Partiendo del directorio raíz del proyecto, todos los archivos relacionados con el kernel y el scheduler se encuentran en el directorio **`kernel`**, donde se encontrarán los siguientes archivos:

kernel\picix.h: Encabezado que todo proceso debe incluir.

kernel\picixkernel.h: Definiciones privadas, no es necesario que ningún proceso las incluya.

kernel\picixkernel.c: Cuerpo de las funciones explicadas en este capítulo, y otras adicionales de utilidad.

4. Diseño del sistema de Archivos

4.1. Introducción

Los sistemas de archivos se crearon como abstracciones para acceder cómodamente a los datos. Esta definición es genérica ya que en este caso, se desea crear un sistema que no sólo sirva para acceder a los medios de almacenamiento existentes en el Microcontrolador, sino utilizar este sistema de archivos para acceder a cualquier fuente de datos, sean los puertos de comunicaciones, conversores ADC, etc., siguiendo la filosofía de Unix de "todo es un archivo".

¿Qué utilidad presta a este proyecto de Firewall? Aunque no es absolutamente necesario para la finalización, un sistema como éste siempre aumenta la calidad final del software, así como su flexibilidad.

> Nota: En la versión final del sistema, se excluye el código de sistema de archivos por bloques, debido a que su utilidad en este caso, no superaba la complejidad que añadía al proyecto, pero su desarrollo se finalizó y es completamente funcional.

4.2. Tipos de Dispositivos

Se debe hacer una última distinción, ya que si se quiere acceder utilizando las mismas funciones `open()`, `read()` y `write()` a un dispositivo como es una interfaz serie RS-232 y a memoria de almacenamiento como la EEPROM on-board, existe una gran diferencia:

Debido a restricciones tecnológicas, la memoria de almacenamiento EEPROM o memoria Flash suele accederse mediante bloques, o sea, no se puede borrar o escribir un carácter en particular, sino que el acceso se realiza en cantidades especificadas por el fabricante, por ejemplo, la memoria Flash del Microcontrolador debe escribirse por bloques de 64 bytes. Por otra parte, es mucho más práctico tener un sistema de archivos sobre una lista o árbol de bloques, que sobre un espacio lineal, ya que de lo contrario surgen problemas al tratar de cambiar el tamaño de un archivo, o de borrarlo. Como conclusión, este tipo de medio debe pasar por un proceso de "formateo" antes de poder ser utilizado como almacenamiento, y serán denominados "**Dispositivos de Bloques**" (ver Tanenbaum:1992 para una descripción detallada).

Por otra parte, la idea de "formatear" un dispositivo como una Interfaz RS-232 es absurda, aunque si se puede utilizar el concepto de `open()` como el de abrir el dispositivo, `read()` como leer caracteres de él, y `write()` como el de enviar caracteres. Por lo que éste tipo de dispositivos debe poder accederse sin realizar previamente ningún tipo de

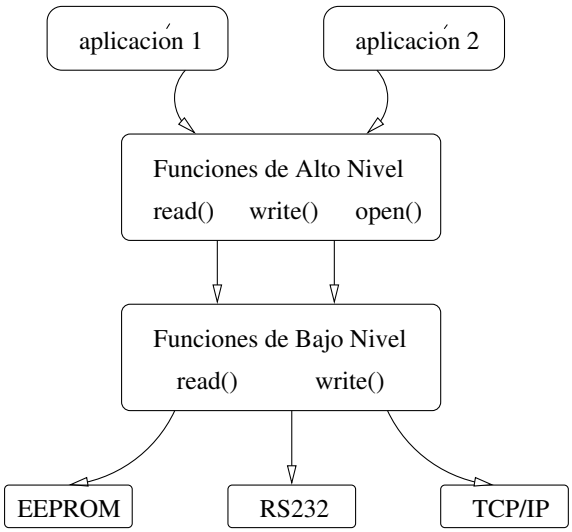


Figura 4.1.: Diseño arquitectónico del sistema de archivos

formateo previo, y serán denominados, para mantener una relación con el sistema Unix, como "**Dispositivos de Caracteres**". Por esto mismo, los dispositivos de caracteres no pueden contener archivos en su interior, sino que sólo se puede leer y escribir caracteres de ellos[1].

> *Nota:* Que un dispositivo sea de caracteres, no significa que obligatoriamente deba accederse al mismo de a un carácter. El dispositivo puede tener restricciones que limitan su acceso por bloques arbitrarios de información. Lo que diferencia un tipo del otro es que los dispositivos de bloques deben ser formateados, y por ello están habilitados para contener archivos y directorios, mientras que los dispositivos de caracteres sólo representan a un sólo archivo.

Para obtener una independencia de dispositivos se puede seguir otra vez, el patrón de diseño por capas (figura 4.1). Actualmente sólo uno de los sistemas de archivos (la segunda implementación) cumple fielmente con este diseño, ya que la primer versión, que se explica a continuación, no posee una estructura de capas definida.

4.3. Primer diseño

El primer diseño, denominado "tfs" por "tiny file system", ponía énfasis en que su implementación sea la más simple y pequeña posible, por lo tanto:

1.No estaba orientado a Bloques.

2.No soporta directorios ni subdirectorios.

3.La implementación de sólo-lectura es muy sencilla, mientras que la escritura es compleja.

[1]Esta definición es propia de este sistema operativo y puede cambiar en otro sistema.

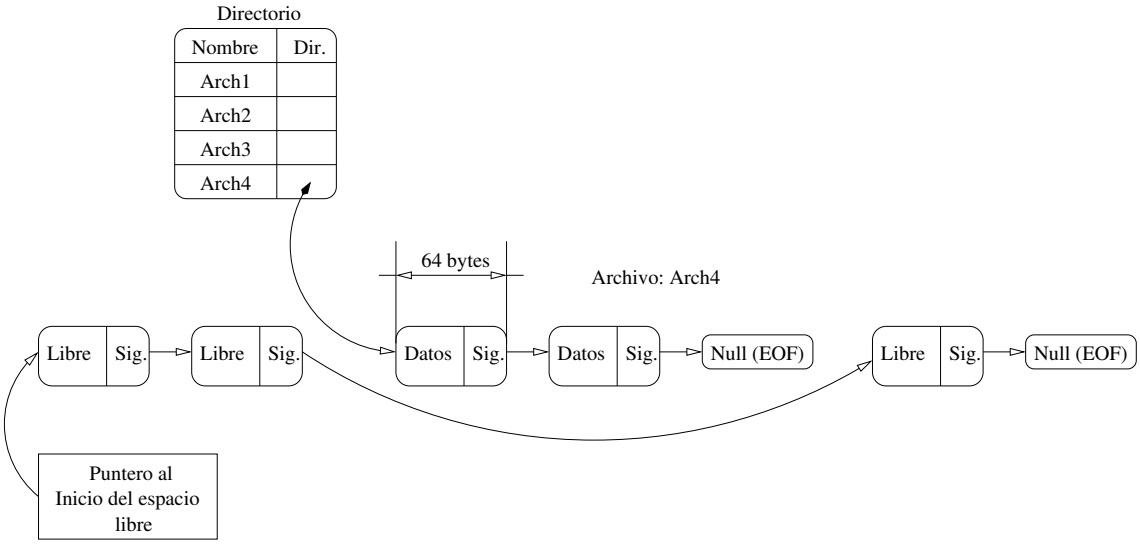


Figura 4.2.: Estructuración de un Archivo y el espacio libre.

Como ventaja, se puede nombrar su pequeño espacio en ROM (1Kb aproximadamente) y la posibilidad de poder contener archivos binarios con código ejecutable, ya que el código no se modificaba al ser almacenado sobre un dispositivo como la memoria Flash del microcontrolador.

La estructura de cada archivo era la siguiente:

- `[unsigned long size][unsigned char type][name][zero][data]`

Y los archivos se apilaban en el medio de almacenamiento como sigue:

- `[file1][file2][file3][free space]`

Como puede verse, la estructura es muy sencilla, pero surgen problemas a la hora de eliminar o cambiar el tamaño de un archivo, ya que se requiere que todos los archivos adyacentes se muevan a un lado o hacia el otro en la memoria, aunque el acceso en sólo-lectura es muy rápido y eficiente.

Este sistema fue implementado casi en su totalidad y se halla en el archivo *tfs.c*, aunque su desarrollo se abandonó por la segunda versión, más compleja aunque mucho más flexible.

4.4. Segundo diseño

El segundo diseño denominado "*tfs2*", está orientado a bloques, o "inodos" y recuerda al sistema de archivos de Unix, aunque en lugar de utilizar un árbol de inodos, se utiliza una lista, por lo que es propenso a la fragmentación y está optimizado para el acceso secuencial a los archivos. La estructura puede verse en la figura 4.2 y se cumplen los siguientes enunciados:

- Al formatear un sistema de archivos, se crea una lista con los inodos libres (en este caso, estarán todos libres).

Listado 4.1: Estructura FS-OP

```
typedef struct

{

void (*read) (unsigned char *buf,unsigned int dst,unsigned char
    len);

void (*write)(const unsigned char *data,unsigned int dst,unsigned
    char len);

void (*info) (unsigned int *max_blocks,unsigned char *about);

} FS_OP;
```

- Cada archivo está formado por una lista de inodos, no necesariamente adyacentes.
- Cada directorio es un archivo normal, que puede ser accedido con las funciones `open(), read() y write()` (y de hecho así se acceden internamente), con el siguiente formato:

```
[char file1[13]] [unsigned int fileStartAddress]
[char file2[13]] [unsigned int fileStartAddress]
...
```

O sea, el largo máximo del nombre del archivo es de 13 caracteres, debido al largo del array `file[]`, y la razón de este número es poder acomodar 4 entradas de archivos por cada bloque, que tiene 64 bytes.

El formato de la dirección de inicio del archivo es la clave para la flexibilidad del sistema de archivos, el entero de 16 bits se divide como sigue:

- `[ 1 bit device type] [2 bits device number] [13 bits startAddress]`

O sea, mediante este método, un archivo puede apuntar a cualquier tipo de dispositivo, como se describe en la próxima sección.

4.4.1. Drivers de Bajo Nivel

Cada driver (manejador) de bajo nivel, debe proveer 3 funciones:

- `read()`: Lee bytes del dispositivo hacia un buffer en memoria.
- `write()`: Escribe bytes desde la memoria al dispositivo.
- `info()`: Devuelve información acerca del driver (nombre, etc.).

Las funciones de mayor nivel las utilizaran para crear la estructura de archivos que se requiera. Cuando se necesite implementar un driver para un dispositivo específico, se debe programar estas tres funciones y luego acomodarlas en una estructura definida como

`FS_OP`. El sistema mantiene un arreglo de punteros a estructuras de este tipo denominado `fileOp[]`, para poder ubicar el driver adecuado mediante su índice. La estructura `FS_OP`, junto con la definición completa de las operaciones de bajo nivel pueden verse en el listado 4.1.

Se realizaron dichas funciones, de las que se encontraran ejemplos de drivers para la memoria Flash, EEPROM, Serial I/O y TCP/IP[2] en el archivo "*`tfs2_ll.c`*".

Ahora, los tres bits más significativos de cada dirección en el sistema de archivos, es en realidad el índice en el array de funciones `fileOp[]` del conjunto de funciones que debe utilizarse, o sea, cada dirección nos está diciendo con cual función se debe leer o escribir, por ejemplo, si nuestra dirección es "direc", el bloque se podría leer de ésta manera:

- `fileOp[direc>>13].read(buffer,direc,10);`

Por lo que se oculta si se está leyendo de la memoria Flash, EEPROM o estamos leyendo caracteres de una conexión TCP/IP.

De hecho, este sistema se utiliza en todas las direcciones que maneja el sistema de archivos, como las direcciones de inicio y de bloque siguiente, por lo que el sistema de archivos contempla la posibilidad de que un archivo se extienda sobre varios medios de almacenamiento, por ejemplo, los primero 64 bytes de un archivo pueden estar en memoria Flash y el resto en EEPROM.

4.4.2. Drivers de alto nivel

Se proveen muchas funciones para que sean utilizados por aplicaciones de usuario, la mayoría se corresponden con la biblioteca "`stdio`" del estándar ANSI de C, aunque todavía no son completamente compatibles con dicho estándar, entre ellas podemos nombrar:

Formatea el sistema de archivos:

- `void mktfs(char fsType);`

Abre para lectura, escritura o creación un archivo dado:

- `char open(const char *path, char mode);`

Cierra un archivo dado:

- `char close( char fd);`

Lee datos del archivo abierto:

- `char read( char fd, char *buf, char qty);`

Guarda datos en el archivo:

[2]La Pila TCP/IP a la que se dedica el capítulo 7, se accede mediante el sistema de archivos, como en la mayoría de los sistemas operativos modernos.

- `char write( char fd,const char *data, char qty);`

Mueve el puntero interno del archivo:

- `char seek( char fd,int delta);`

Elimina un archivo del sistema de archivos:

- `char erase(const char *name, char unlink);`

El tamaño de bloque está fijo en 64 bytes, para que se corresponda con el tamaño mínimo de bloque de la memoria Flash del microcontrolador PIC-18, por lo tanto, se puede calcular el tamaño máximo de un sistema de archivos, de ésta manera:

Como puede verse en la sección 4.4, existen un total de 13 bits disponibles para direccionar el bloque de inicio del archivo. Como $2^{13} = 8192$, se puede acceder a 8192 bloques.

Ahora bien, como cada bloque tiene 64 bytes cada uno, el tamaño máximo del sistema de archivos es de $8192 * 64 = 512K$, aunque el espacio útil sería 496 Kb (2 bytes de los 64 guardan la dirección del bloque siguiente). Se considera que 500Kb es un máximo respetable para un sistema de archivos de estas características, al menos por ahora.

4.5. Salida y entrada estándar

Siguiendo la costumbre de la mayoría de los sistemas operativos, al iniciar el sistema se abren dos archivos, señalizados por STDIN y STDOUT, apuntando al principio al driver de entrada/salida serie, por lo que cualquier escritura o lectura hacia esos archivos efectuará una salida hacia el terminal o una lectura del teclado. se proveen funciones para reemplazar estos archivos por otros, de manera que la entrada y salida estándar puedan ser redireccionadas a cualquier otro archivo o dispositivo.

Para que esto funcione, el sistema de archivos debe proveer la función `putch(char b)`, que en realidad sea redefinida como `fputch( STDOUT , b )`, y lo mismo con `getch()`.

La función `fputch()` se implementó con un pequeño buffer configurable, ya que escribir muchos datos de a un byte a la vez puede llegar a dañar la memoria Flash del dispositivo, que tiene un limite de 100000 grabaciones, sin embargo, el buffer debe poder anularse, ya que cuando `fputch()` apunta a la salida serie, es deseable que los caracteres se impriman en la pantalla tan pronto como sea posible[3].

4.6. Conclusiones y mejoras

El sistema de archivos “tfs2” (que se reitera, no fue utilizado en el sistema final pero fué implementado completamente) es una parte del sistema operativo realizada para soportar al sistema de firewall, pero al igual que el kernel es un módulo independiente, y

[3] En la versión final, se optó por un buffer separado de entrada y otro de salida para los drivers de bajo nivel (Específicamente los drivers de TCP).

puede ser utilizado en cualquier otro proyecto que requiera una abstracción de los dispositivos de entrada/salida y almacenamiento. En realidad, ni siquiera necesita compilarse sobre la arquitectura PIC-18. Ocupa unos 6 Kb de código y 80 bytes de memoria RAM (depende la cantidad máxima de archivos abiertos permitidos).

Entre las mejoras que pueden realizarse:

1. Optimizaciones del código para acceso a directorios
2. Flexibilización del formato de los directorios (tamaño del nombre ilimitado, etc.)
3. Realización de drivers de bajo nivel para todos los dispositivos del PIC
4. Implementación de un tipo de archivo binario especial, que permitiría la ejecución de código directamente desde la memoria Flash.
5. Mejora del sistema de buffers interno, para aumentar la perfomance.

4.7. Ubicación de Archivos

Partiendo del directorio raíz del proyecto, todos los archivos relacionados con el sistema de archivos se encuentran en el directorio `fs`, donde se encontrarán los siguientes archivos:

fs\tfs2.h: Encabezado que todo proceso debe incluir si desea utilizar las funciones del sistema de archivos.

fs\tfs2.c: Cuerpo de las funciones de alto nivel del sistema de archivos.

fs\tfs2_ll.h: Definiciones privadas de los drivers de bajo nivel, no es necesario que ningún proceso las incluya.

fs\tfs2_ll.c: Cuerpo de los drivers de bajo nivel.

5. Shell (Intérprete de comandos)

El Shell o Intérprete de comandos, es una aplicación de usuario y no suele formar parte del sistema operativo, pero es incluida dentro del sistema por estar dentro del llamado "software de soporte" del Firewall. Equivaldría al archivo "*CMD.EXE*" de Windows o "*bash*" de los sistemas Unix.

La estructura es sencilla: es una función a la cual se le asigna un proceso del kernel, que lee comandos de STDIN, y envía la salida a STDOUT o a un archivo.

Otra de las funciones que podría cumplir es la de procesar archivos de comandos y ejecutar archivos binarios[1].

Por ahora los comandos se agregan en tiempo de compilación en una estructura como la se observa en el listado 5.1.

En este listado puede verse cómo se completa la estructura `TCOMMAND` con el nombre del comando, la función a invocar, y una pequeña cadena que representa la ayuda. El shell también cumple las funciones de separar los parámetros que recibe cada comando, y redirigir la entrada o la salida según corresponda, mediante los operadores ">" ó "<".

5.1. Llamada a comandos

Podría llegar a pensarse que el shell es en realidad una función que llama a otras funciones, como lo hace cualquier aplicación. Sin embargo, es importante notar que en esta versión, la función de shell no invoca a los comandos directamente, sino que llama a la función `OS_Replace()`[2] para que el sistema operativo elimine los recursos utilizados y reemplace el proceso `shell()` por la función correspondiente al comando, que recibe una pila de hardware y software vacía, y por lo tanto tiene más memoria libre para utilizar. Esta opción se ejemplifica en la figura 5.1 b), donde mediante un solo hilo de ejecución se ejecutan ambas funciones secuencialmente.

Otra opción, como se muestra en el punto a) de la figura 5.1, hubiera sido crear un proceso diferente para cada comando. Esta es la aproximación utilizada por sistemas mayores, y aunque consume mucha memoria (demasiada para este proyecto) tiene la ventaja de poder lanzar comandos que se ejecuten en segundo plano. El scheduler está en condiciones de realizar tal procedimiento, aunque se consumiría mucha más memoria.

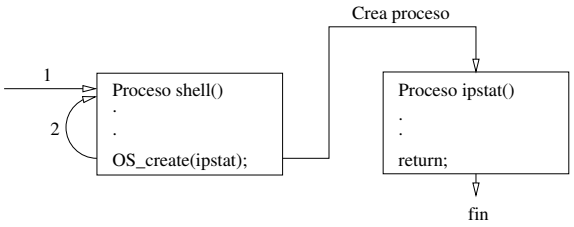
[1] Éstas son características futuras en nuestro sistema, ya que necesitan soporte por parte del sistema de archivos para funcionar.

[2] Provista por el servicio de scheluder del sistema operativo.

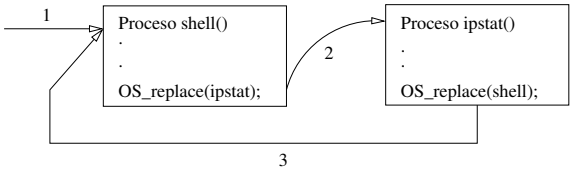
Listado 5.1: Estructura de comandos del shell

```
typedef struct
{
const rom unsigned char *Nombre;
void (*proc)(void);
const rom unsigned char *Ayuda;
} TCOMMAND;


const rom TCOMMAND comando[] =
{ {"man",ayuda,"Imprime la ayuda"},
{"ver",version,"Imprime la versión"},
{"top",top,"Imprime los procesos"} ...
luego:
void ayuda(void) { ... }
void version(void) { ... }
void top(void) { ... }
```



(a) Dos hilos



(b) Un solo hilo

Figura 5.1.: Opciones de llamada a comandos.

Se codificaron varias opciones en la línea de comandos para comodidad del usuario, tales como un historial[3] y se completan automáticamente los comandos al presionar la tecla "*TAB*".

A continuación se listan los comandos que están disponibles en la versión final del shell. Pueden dividirse en tres grupos:

1. Información acerca del sistema:

   man: Imprime la ayuda

   ver: Imprime la versión

2. Información y manejo de los procesos:

   ps: Imprime información acerca de los procesos del sistema

   kill: suspende o reinicia un proceso

   renice: modifica la prioridad de un proceso

3. Manejo y estadísticas del firewall:

   fw: [port] [-c] :Modifica opciones de firewall

   arp: [-c]: Muestra la tabla ARP. (Para más información, ver sección 7.4 en la página 72)

[3] El Sistema es muy limitado en cuanto a memoria RAM, por lo que el historial se limita a una sola linea, y solamente los primeros 16 caracteres de la misma!

ip: [ip] [netmask] [router]: Configuración IP

mac: [addr]: MAC Address

ipstat: [-a]: Muestra estadísticas de red

saveconf: : Guarda la configuración actual

5.2. Ubicación de Archivos

Todo el sistema descrito se encuentra en un solo archivo, denominado `cshell.c` y su correspondiente encabezado `cshell.h` ubicados en el directorio raíz del proyecto. No todos los comandos se encuentran implementados en esos dos archivos, ya que se consideró más apropiado colocarlos junto a los módulos que manejan, por ejemplo, el comando `ip` que maneja la configuración de la pila TCP/IP, se encuentra implementado en un archivo de código fuente localizado junto a los demás archivos de la pila.

6. Interfaces de Red

Las interfaces de red son uno de los componentes básicos del sistema, ya que permiten al microcontrolador acceder a los datos que circulan en una red de computadoras. Los datos se suelen agrupar en forma de paquetes y circulan a alta velocidad. Hacen falta dos interfaces, ya que el microcontrolador debe intercambiar de manera transparente todos los paquetes autorizados de una red hacia la otra, como se ve en la figura 1.1 en la página 14.

6.1. Hardware

Desde un principio se decidió utilizar la tecnología Ethernet de red local, debido a que es la utilizada en la gran mayoría de las redes actuales (ver Stallings:2000), por lo que el sistema se diseñó alrededor de esta tecnología y sus requerimientos, que son muy importantes, como se ve en el capítulo 2.

Existen varios circuitos controladores de Ethernet 10Base-T, algunos con más de 10 años en el mercado, entre ellos:

- 8390 (También llamado NE2000, que luego se convirtió en un estándard para 10Base-T)
- 3com 3c5X9
- Realtek 8039
- Cirrus Logic CS8900A
- Y muchos otros más.

El componente seleccionado fue el integrado CS8900A de Cirrus Logic, controlador "*single chip*" de Ethernet 10Base-T / AUI, que posee un excelente soporte de software. Es relativamente sencillo de programar, y existe código fuente disponible tanto en el lenguaje "C"[1] como en código ensamblador RISC utilizado en los PIC.

Los drivers fueron desarrollados desde cero, aunque basados en las recomendaciones del manual y otros drivers de ejemplo. Se utilizó el lenguaje C, aunque para la versión final se optimizaron las funciones críticas con porciones en código ensamblador.

El formato del circuito integrado CS8900A es de montaje superficial TQFP y más 100 pines, por lo que se necesitan herramientas especiales de montaje. Sin embargo, puede

[1] El sistema operativo Linux tiene uno de los mejores drivers para éste modelo, según el propio manual del CS8900A.

adquirirse en forma de módulo ya preparado para su inserción a un microcontrolador de 8 bits, lo que facilita mucho la tarea del armado del circuito. Se optó por adquirir un par de dichos módulos (uno para cada extremo del firewall), que aunque poseen un precio muy superior al precio de venta del circuito integrado, este último no se vende en cantidades individuales, sino por un mínimo de 500 en algunos comercios estadounidenses. Una representación de los módulos, junto con la explicación de cada uno de los componentes puede verse en la figura 2.2 en la página 21.

6.2. Inicialización

Ambas interfaces deben inicializarse para poder funcionar, ya que son sistemas complejos que poseen procesamiento y memoria propia. Se debe seguir un determinado proceso denominado *"reset"* en el cual se configuran todas las variables del dispositivo.

6.2.1. Modo promiscuo

Una de las variables más importante a configurar es la que maneja el modo '*promiscuo*' que merece mayor explicación: Una red se compone generalmente de muchas computadoras, que se comunican entre sí. Al intercambio de información entre dos computadoras se lo llamará *conversación*. En una red pueden existir simultáneamente múltiples conversaciones, algunas de las cuales pueden ser privadas, por lo que no es deseable que ninguna computadora escuche tales conversaciones a excepción de los participantes. Al ser Ethernet una red cuyo medio de comunicaciones puede estar compartido entre todos los nodos, resulta inevitable que la información se envíe a todos. Como puede verse en la figura 6.1, el nodo A le envía un paquete al nodo C, pero debido al funcionamiento de la red, el *hub* (repetidor) también realiza una copia de la información para el nodo B, aunque éste no participe en la conversación. Para brindar privacidad en este tipo de redes, cada interfaz está configurada de manera que, aunque reciba los paquetes de toda la red, sólo aceptará aquellos que le fueron enviados a ella misma. Ahora, prácticamente todas las interfaces de red Ethernet permiten anular este filtro, para permitir escuchar los paquetes de toda la red y no solamente los enviados a ella. Para realizar esto, se debe colocar la red en un modo especial, denominado modo '*promiscuo*', en el que aceptará todos los paquetes de la red, y se los pasará al sistema operativo. Este modo es utilizado en programas denominados '*sniffers*' que espían una red para extraer información. El sistema de firewall también necesita de esta opción activada, o de lo contrario sólo aceptaría paquetes enviados al mismo firewall, y rechazaría cualquier otro paquete que se envía de una red hacia la otra, por lo que al inicio, se debe configurar ambas interfaces en este modo.

6.2.2. Dirección MAC

Luego de la lectura anterior, puede intuirse que cada interfaz necesita un identificador único dentro de una red, para que todos los nodos puedan ubicarse individualmente. Este identificador se denomina dirección de hardware o dirección MAC (Medium Access Control, control de acceso al medio), ya que es la dirección que utiliza la capa MAC de la red. Esta dirección es un número entero de 48 bits que identifica de manera única a la interfaz en cualquier red que se utilice. Si en una red llegaran a existir dos placas con

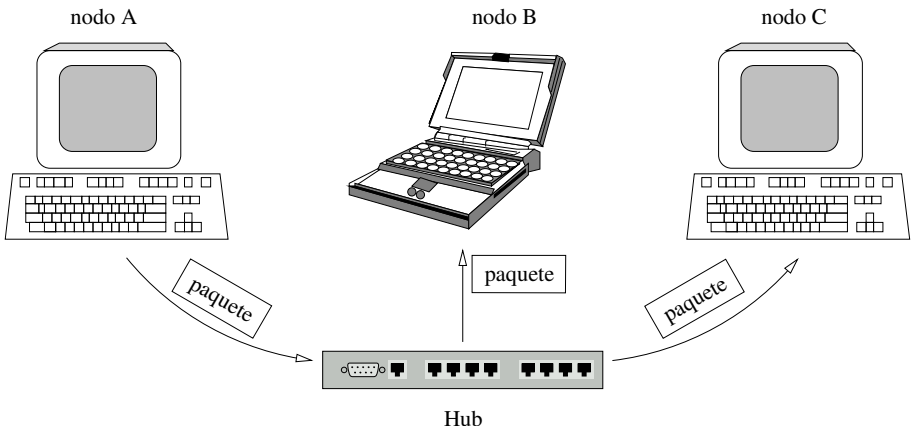


Figura 6.1.: Comunicación en un medio compartido

la misma dirección , el funcionamiento general se vería gravemente afectado, por lo que la dirección MAC es asignada por una entidad global, que asegura que cada fabricante de interfaces Ethernet obtendrá un conjunto de identificadores únicos para asignar a los productos que fabrica.

Esta dirección, que siempre se consideró inmodificable, es utilizada como una “huella dactilar” para identificar a todo sistema de red. Sin embargo, durante la construcción del dispositivo de firewall se llegó a un sistema que permite modificar esta dirección, debido al siguiente funcionamiento de las interfaces:

Como muchos otros controladores Ethernet, el chip CS8900A permite almacenar la configuración en una memoria especial EEPROM, que debe conectarse de una manera especial a sus terminales. De esta manera, la interfaz puede recordar su configuración para que no se tenga que setear en cada inicialización. El fabricante guarda en esta EEPROM muchas variables de configuración, entre ellas la dirección MAC, que de esta manera pasa a ser fija, ya que no se permite la alteración de esa variable en la memoria EEPROM.

Este es el funcionamiento en la gran mayoría de las interfaces Ethernet, pero si dicha memoria de configuración llegara a faltar, el circuito integrado está obligado a solicitar su configuración a través de sus puerto de entrada/salida, o sea, debe configurarse todas las variables mediante los drivers.

Como se ve en la sección 2.2.2 en la página 21 (componentes), el módulo Ethernet no posee la EEPROM de configuración, por lo que es tarea del driver de bajo nivel asignar dicha dirección MAC, pudiendo cargar en este lugar cualquier número. De todas maneras, al setear las interfaces en modo promiscuo, la dirección MAC no se utiliza casi nunca, ya que todos los paquetes son aceptados, sean para la dirección del firewall o no.

La configuración final setea ambas interfaces con una sola dirección MAC, configurable por el usuario, que es necesaria sólo en el caso de utilizar la configuración vía Web, ya que el sub-sistema TCP/IP necesita poseer una dirección única de hardware. Se debe aclarar que al permitir la modificación de la dirección MAC, el sistema está violando los estándares de Ethernet.

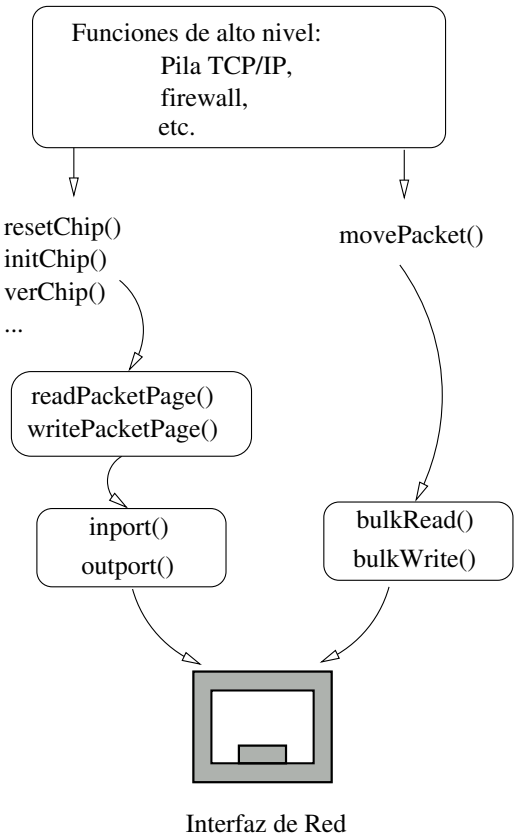


Figura 6.2.: Estructura en capas de los drivers

6.3. Drivers

Según el manual del CS8900, éste se controla por medio de un sistema denominado "*packetPage*" para acceder a los múltiples registros Internos. Este sistema implica depositar ciertos datos utilizando el bus[2] de direcciones y el bus de datos[3] (para más detalle sobre el bus, consultar la sección 2.6 en la página 31), para poder acceder a los registros de configuración especial de la interfaz Ethernet. Se implementaron varias funciones que corresponden a diferentes niveles, algunas más cercanas al hardware, mientras que otras se abstraían completamente del mismo, como se ve en la figura 6.2.

En contacto directo con el bus de datos y direcciones están las funciones `outport()` e `inport()`, que sólo leen o envían bytes modificando adecuadamente los valores de los puertos y la dirección de los mismos. Se recuerda que el bus de datos es bi-direccional, por lo que electrónicamente se compone de 8 entradas/salidas, y el bus de direcciones se compone de sólo 4 salidas, y no es bidireccional, sino que el microcontrolador es el único dispositivo que lo maneja.

Las funciones `outport()` e `inport()` son la base para la capa de software superior, conformada por las funciones `readPacketPage()` y `writePacketPage()`, que efectivamente escriben y leen registros del dispositivo. Subiendo un nivel más, se hallan las fun-

[2]Puerto A del PIC-18.
[3]Puerto D del PIC-18.

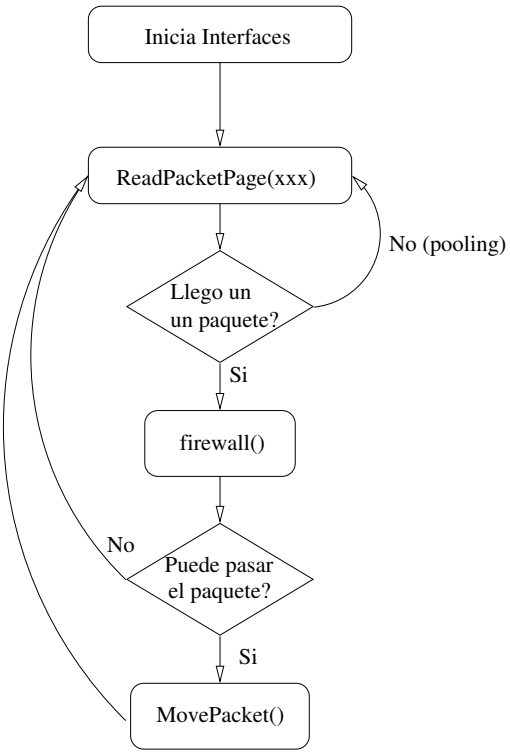


Figura 6.3.: Diagrama en bloques del funcionamiento general del firewall

ciones de (relativo) alto nivel, que inicializan y resetean el chip, utilizando una secuencia de llamadas a las funciones de manejo anteriores.

Un caso especial son las funciones para mover paquetes entre las interfaces. Dichas funciones deben estar optimizadas al máximo y para ello deben acceder directamente al hardware. En la figura 6.2 pueden observarse con los nombres de `bulkRead()` y `bulkWrite()`, que leen y escriben datos en bloques de manera muy rápida entre las interfaces y el microcontrolador. Estas funciones son utilizadas por la función `movePacket()`, que agrega el protocolo necesario para recibir y enviar el paquete por ambas interfaces. Esta última función se utiliza para mover un paquete entre las interfaces, luego por supuesto de haber pasado por la función de filtro o `firewall()`, en el archivo `firewall.c`, que es el núcleo del sistema. El funcionamiento descrito se ilustra en la figura 6.3, donde se muestra el proceso de pooling que se realiza y se muestran los pasos a seguir en caso de que haya llegado un paquete.

6.4. Ubicación de Archivos

Toda las funciones concernientes a las interfaces de red se encuentran en un solo archivo, denominado `nic.c` y su correspondiente encabezado `nic.h` ubicados en el directorio **`drivers\nic`** del proyecto.

En el archivo `cs8900.h` ubicado en el mismo lugar que los anteriores, pueden observarse definiciones correspondientes a las direcciones específicas del integrado CS8900A de Cirrus Logic.

7. Pila TCP/IP

7.1. Introducción

Uno de los componentes más complejos del sistema es la pila TCP/IP, que implementa este conjunto de protocolos estándar de comunicaciones. Este componente es una parte muy importante de todo sistema operativo de redes, tal como Unix o Windows 2000, y provee servicios variados, tales como comunicaciones orientadas a conexión (TCP), no orientadas a conexión (UDP), aparte de mantener una serie de servicios de bajo nivel utilizado por la red, tales como los mensajes ICMP.

Una característica importante de estos sistemas es la interfaz que se expone para que los programadores la utilicen, ya que de esto dependerá la complejidad y flexibilidad del software generado. Con el tiempo se ha llegado a un consenso general para adoptar la interfaz de "sockets", que se estudia en cursos de programación orientada a redes.

Teniendo en cuenta la memoria total del sistema (32 Kbytes) es deseable que la pila TCP/IP no ocupe más de 5 Kb de código, por lo que es preciso acotar los servicios prestados, tratando al mismo tiempo de no complicar demasiado la interfaz al programador.

Ya que una pila TCP/IP es un componente muy utilizado también en sistemas embebidos[1] (que no sólo se comunican con interfaces Ethernet), existe mucho software disponible en el mercado, algunos de libre distribución, aunque la mayoría apunta a procesadores de 16 o 32 bits, con un tamaño de código mínimo de 30-40 Kb, lo que excede la capacidad del microcontrolador, que posee un máximo de 32 Kbytes de memoria de programa.

La única opción disponible a la fecha, era UIP de ADAMS DUNKELS, distribuido gratuitamente bajo licencia GNU[2]. Luego de la finalización del proyecto, se tomó conocimiento que la empresa Microchip también distribuye de manera gratuita una pila TCP/IP que corre en los procesadores PIC-18, con una estructura muy similar a UIP, aunque es un poco más completa en cuanto a los protocolos que soporta.

7.2. UIP

UIP es un software muy interesante, realizado en su totalidad en ANSI C, posee un tamaño de Código muy pequeño, ya que compila a unos 4 Kb de Código RISC, y provee las siguientes opciones:

[1] Se denomina embebido a cualquier sistema que posea un computador programable como uno de sus componentes, Ej.: Teléfonos celulares, sistemas de misiles, telares automáticos, etc.

[2] La licencia GNU permite distribuir el software de manera gratuita junto a su código fuente, con la condición de que toda modificación debe a su vez ser publicada. En este sentido es la más restrictiva de las licencias de software de libre distribución.

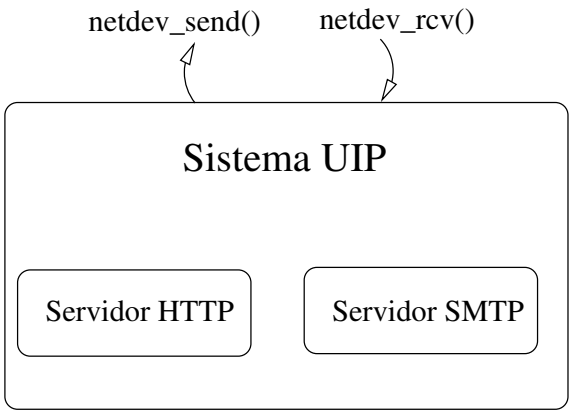


Figura 7.1.: Estructura de una aplicación utilizando UIP

- Múltiples conexiones TCP simultáneas.
- Capacidad de generar conexiones (Ser un cliente), y de aceptar conexiones (ser un servidor)
- Generación de múltiples estadísticas de uso.
- Servicio ICMP de “Echo” (respuesta al comando “ping”)

Sin embargo, el software supone una estructura muy especial de la aplicación. Específicamente, supone que la aplicación que utilice UIP es una máquina de estados, y dicho “estado” es un conjunto de datos que debe guardarse en una posición de memoria manejada por el sistema UIP[3]. Desde el punto de vista del código fuente, es evidente la adaptación de UIP para ambientes con bajos recursos, en el sentido que:

1. No utiliza variables automáticas: Todas las variables son globales, por lo que el código no es reentrante, ni está preparado para ejecución paralela.
2. Utiliza extensivamente los saltos incondicionales “goto”: Aunque no se utilizan comúnmente en el lenguaje “C”, UIP ahorra mucho código utilizando este método para evitar llamadas a funciones .

Como se ve en la figura 7.1, la aplicación es en realidad un módulo del sistema UIP, que es una estructura inversa a la utilizada normalmente. Además, los sistemas del tipo “máquina de estados” se utilizan generalmente en microcontroladores que no disponen de un sistema operativo, debido a sus limitaciones. Pero el microcontrolador PIC-18 dispone de la potencia suficiente para implementar un sistema basado en capas.

[3]De ésta manera se simula la concurrencia de múltiples conexiones, al ser el propio UIP el encargado de manejar dicho estado.

7.3. Pila TCP/IP desarrollada

Basado en el trabajo de Adam Dunkels, se procedió a la creación de una pila TCP/IP propia, como puede apreciarse en la figura 7.2, muy diferente en su funcionamiento ya que forma parte del sistema operativo como un servicio de comunicaciones. De hecho, el acceso a una conexión TCP se realiza mediante el sistema de archivos, utilizando un sistema muy similar a los "sockets" utilizados en sistemas operativos comunes. Las funciones utilizadas son las siguientes:

- `unsigned char listen(unsigned int port);`

El equivalente a las funciones `socket(),listen() y accept()` de Posix[4], abre un puerto TCP y devuelve un descriptor de archivos, del mismo tipo que devuelve la función `open()`.

- `unsigned char TCPstatus(unsigned char fd);`

Devuelve información extendida acerca del estado de la conexión, ya que a diferencia de un archivo, que puede estar abierto o cerrado, una conexión puede estar en varios estados, entre ellos:

LISTEN: Esperando conexión

SYN_RCV: Esperando confirmación de conexión

ESTABLISHED: Establecida.

CLOSED: Cerrada. Debe llamarse a `TCPClose()` para liberar todos los buffers y volver a abrirse mediante otra operación `listen()`.

- `void TCPsetInteractive(unsigned char fd);`

Se utiliza para conexiones que deben ser interactivas, tales como TELNET, ya que como la pila implementada no posee buffers suficientes, mediante esta función se le pide que deseche todos los datos que recibe si no son requeridos.

- `void TCPClose(unsigned char fd);`

Envía todos los datos que estén en el (pequeño) buffer de salida, y libera el descriptor de archivo asociado.

[4]Posix es un estándar que deben cumplir todos los sistemas operativos compatibles con Unix. En teoría, hasta Windows 95 cumple con el estándar Posix.

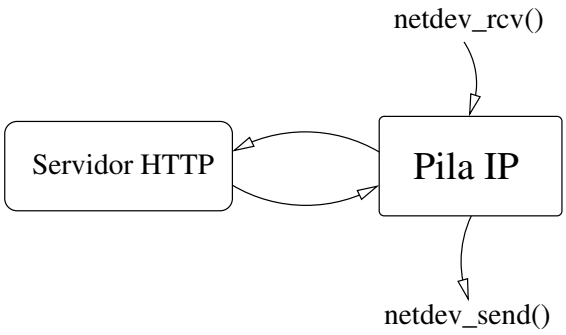


Figura 7.2.: Enfoque adoptado para la Pila TCP/IP desarrollada

Las funciones `netdev_send()` y `netdev_rcv()` son parte de los drivers de bajo nivel, utilizados para la comunicación con las interfaces de red.

Una adición importante de la pila implementada es que, al contrario de UIP, permite la configuración dinámica de sus parámetros, tales como dirección IP, ruta por defecto, máscara de sub-red, etc.. Como se ve en la sección 6, gracias a una característica del hardware, es posible configurar dinámicamente la dirección MAC, opción que no se encuentra en sistemas normales[5].

7.3.1. Funcionalidad faltante

Una pila TCP/IP es un componente relativamente complejo, cuyo código ronda los 200 Kbytes en una implementación completa[6]. Una implementación que ocupe sólo 5 Kbytes deberá dejar alguna funcionalidad de lado. En este caso, se pueden citar tres características faltantes importantes:

Imposibilidad de iniciar una conexión TCP: Debido a que el sistema nunca necesita realizar esta operación el código necesario no fue implementado, aunque si fuera necesario sería muy sencillo modificar el código actual para permitirlo, ya que sólo se debe implementar el *hand-shaking*[7] inicial, y una vez establecida la conexión puede utilizarse el código existente para manejo de conexiones TCP.

Soporte de UDP: Este protocolo tampoco fue implementado, y se requeriría de un esfuerzo adicional, que quizás elevaría el tamaño de código a niveles inaceptables.

Hand-shake de finalización completo: Al cerrar una conexión TCP no se sigue completamente el protocolo de finalización, aunque esto no presenta problemas de ningún tipo, al menos con el uso que se le da en el proyecto actual.

En el futuro podrían llegar a implementarse estas características, pero en la versión final fueron dejadas de lado con el objeto de acotar la duración del proyecto.

[5] Por una buena razón, ya que variando la dirección MAC podrían llegar a realizarse multitud de ataques o acciones mal-intencionadas de manera indetectable.

[6] fuente: kernel de Linux

[7] El *hand-shake* es un conjunto de pasos que deben seguir dos nodos en una red para iniciar o cerrar una conexión TCP.

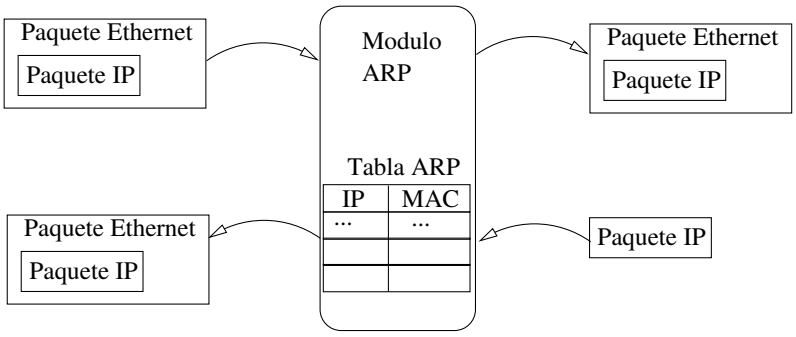


Figura 7.3.: Funcionamiento del Protocolo ARP.

7.4. ARP

Un protocolo indispensable en ambientes de red local tales como Ethernet, es ARP, que básicamente traduce direcciones IP a direcciones de hardware MAC.

El código ARP utilizado es en un 80 % el original de UIP, con algunas modificaciones. Especialmente fueron corregidos muchos errores que en realidad, inutilizarían el código de UIP de usarlo sin modificaciones. Por ejemplo, en un paquete ARP de respuesta a una petición, se incluye sólo el primer byte de los seis que conforman la dirección MAC. Esto no sólo es un gran error, sino que inutiliza todo el código. Sin embargo, como el resto del código funciona bastante bien, es bastante extraño que la distribución de UIP, versión 0.6, contenga un error de estas características. Se comunicaron los errores al autor, ADAM DUNKELS y eventualmente serán corregidos.

La implementación del protocolo ARP es sencilla, ya que sólo debe mantenerse una tabla (ver figura 7.3) que asigne direcciones IP a direcciones de hardware, que se actualiza mediante paquetes especiales denominados “Peticiones ARP” o por otros métodos (ver Comer:1995). Cuando se necesita enviar un paquete hacia algún nodo en la red local, se consulta la tabla para averiguar su dirección de hardware. Si dicha dirección no está en dicha tabla, se debe enviar un paquete de petición y agregar la dirección retornada como una entrada en la tabla. Si no hay lugar, se eliminará la entrada más antigua.

La tabla tiene un tamaño configurable en tiempo de compilación, lo que significa que no puede alterarse una vez compilado el sistema. De este tamaño dependerá la cantidad de paquetes de “peticiones ARP” que se envían, ya que si la tabla es muy pequeña, se deberán enviar multitud de estos paquetes debido a que las entradas serán reemplazadas con mayor frecuencia. Puede pensarse en la tabla como una “memoria caché” de peticiones ARP. En el sistema final, la tabla posee un tamaño de 5 entradas, lo que resultó adecuado.

Debido a que las direcciones de hardware suelen cambiar con frecuencia (Ej. se cambia una placa de red, o se actualiza una PC, etc.) cada entrada en la tabla tiene un vencimiento que al cumplirse implica la remoción automática de dicha entrada. Este vencimiento es de 20 minutos en la mayoría de las pilas TCP/IP, aunque la nuestra tiene un tiempo menor, de 4 minutos, para utilizar contadores de menor tamaño.

La tabla ARP del sistema puede manipularse mediante el comando `arp` , que permite listar la tabla ARP, o bien inicializarla. En una implementación mayor como la existente en Windows 2000 o Linux, se permite agregar entradas estáticas manualmente, eliminar selectivamente algunas entradas, etc.

7.5. Seguridad

Se agregaron ciertas características de seguridad a la Pila que quizás sean de interés al lector.

7.5.1. Campos aleatorios

Existen dos campos que deberían ser generados aleatoriamente para evitar los ataques denominados “Secuestro de sesión TCP” en el que se adivina el número de sesión e identificación IP, y se envían paquetes simulando ser una de las partes comunicantes. Para evitar éste tipo de ataques, los campos “IpID” de todos los encabezados IP y el campo de secuencia inicial de TCP se generan de forma aleatoria.

El campo “IpID” que se utiliza para ensamblar paquetes IP fragmentados, ha tenido un nuevo uso recientemente: nunca se había prestado atención a éste valor dentro del paquete IP, y se implementaba como un contador global en el caso de la pila de sistemas Windows, o cero en el caso de Linux. Teniendo en cuenta éstos datos, es posible identificar cuántos nodos se hallan detrás de un router de enmascarado (Masquerading) o NAT (Network Address Translation), técnicas muchas veces utilizada para compartir una sola conexión IP entre varios computadores, y que se consideraba, ocultaban toda la información acerca de la red protegida. Haciendo este campo un valor aleatorio, efectivamente se impide este tipo de análisis externo.

7.5.2. Rastreo de Pila Activo

El rastreo de pila es una técnica muy potente para averiguar con rapidez cuál es el sistema operativo instalado en el Host, y funciona aprovechando el hecho de que todas las pilas TCP/IP difieren en ciertas características a la hora de responder a ciertos paquetes. La pila implementada se caracteriza por ser muy simple y no responder casi a ningún paquete a excepción de un pedido SYN de TCP.

Se utilizó una de las herramientas más potentes para realizar un rastreo de pila activo contra el dispositivo, para averiguar qué datos se podría obtener sobre él. Se trata del conocido software nmap, que dispone de una técnica de rastreo denominada “tcp fingerprinting” que alude a que dispone de una base de datos de “huellas digitales” producidas por las pilas TCP/IP de multitud de sistemas. A continuación se lista la salida al utilizar el comando “`nmap -O`” contra la dirección IP del dispositivo. Como puede verse a continuación, el firewall está configurado con el IP 192.168.0.5 y puede verse que tiene el puerto 80 (servidor web) abierto.

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Host (192.168.0.5) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.5)
Adding open port 80/tcp
WARNING: RST from port 80 -- is this port really open?
WARNING: RST from port 80 -- is this port really open?
WARNING: RST from port 80 -- is this port really open?
WARNING: RST from port 80 -- is this port really open?
WARNING: RST from port 80 -- is this port really open?
The SYN Stealth Scan took 93 seconds to scan 1150 ports.
For OSScan assuming that port 80 is open and port 1 is closed and
```

```
neither are firewalled
Insufficient responses for TCP sequencing (0), OS detection may be
less accurate
Interesting ports on (192.168.0.5):
(The 1149 ports scanned but not shown below are in state: closed)
Port State Service
80/tcp open http
Remote OS guesses: ComOS - Livingston PortMaster or
U.S. Robotics/3com Total Control system, Lucent Portmaster 4
running ComOS v4.0.3c2
Nmap run completed -- 1 IP address (1 host up) scanned in 101 seconds
```

Obviamente el firewall no se encuentra en la base de datos, por lo que nmap trató de adivinar y lo confundió con otros dispositivos que deben poseer un comportamiento similar.

7.6. Línea de Comando

Se proveen tres funciones para mostrar o manejar la configuración de la pila TCP/IP:

MAC: Se utiliza para mostrar o modificar la dirección MAC de *ambas* Interfaces, ésto se hace así para ahorrar memoria y procesamiento al poder tratar ambas interfaces como una sola, desde el punto de vista de peticiones TCP/IP. El hecho de cambiar ésta dirección implica un reset sobre ambas interfaces, aunque ésta operación es tan rápida que no puede percibirse.

IP: Comando similar a "`ifconfig`" de linux, se debe especificar la dirección IP, máscara de sub-red y router por defecto. Por medio de los parámetros "`up`" y "`down`" se activa y desactiva respectivamente todo el sub-sistema IP. Este comando también acepta los parámetros "`up0`" y "`up1`" que se utilizan para indicar a la pila TCP/IP que acepte paquetes solamente de una de las interfaces e ignore cualquier actividad en la otra, opción muy útil en ambientes inseguros.

ARP: Muestra la tabla ARP, y si se utiliza con la opción "-c" borra todas las entradas de la misma.

7.7. Ubicación de Archivos

Partiendo del directorio raíz del proyecto, todos los archivos relacionados con la pila TCP/IP se encuentran en el directorio **`ip`**. Se mantuvo el prefijo "uip_" a todos los archivos que contuvieran código de la distribución UIP 0.6 de Adams Dunkels.

ip\ip.h: Encabezado que todo proceso debe incluir si desea utilizar las funciones de la pila TCP/IP.

ip\ip.c: Cuerpo de las funciones de la pila TCP/IP nombradas en este capítulo.

ip\uip_arp.h: Encabezado correspondiente a la implementación del protocolo ARP.

ip\uip_arp.c: Cuerpo de la implementación del protocolo ARP.

ip\picix_ip.h: Encabezado con la definición de funciones que relacionan la pila TCP/IP con el resto del sistema, tales como la implementación de los comandos `mac`, `ip`, `arp`, y otras funciones.

ip\picix_ip.c: Cuerpo de las funciones citadas en el archivo anterior

ip\uip_arch.h: Definición de funciones de cálculo de checksum, extraídas de UIP.

ip\uip_arch.c: Cuerpo de dichas funciones.

8. Servidor web (Http)

8.1. Introducción

El sistema de firewall puede configurarse de dos maneras distintas: el método preferido es mediante la línea de comandos, el cuál presenta mayor flexibilidad y se accede a una mayor cantidad de opciones. Este método se detalla en el capítulo 9.

El método secundario que se trata en este capítulo utiliza una interfaz web, más amigable que una línea de comandos para el usuario común. Se recuerda que este tipo de interfaz está basada en el paradigma cliente-servidor y por lo tanto consta de dos componentes principales:

Servidor: Es una aplicación que básicamente envía archivos al cliente utilizando el protocolo http. El código de dicho servidor reside y es ejecutado dentro del microcontrolador.

Aplicación cliente: Se denomina navegador o "browser" y generalmente reside en la computadora desde la cual el usuario desea administrar el Firewall.

Actualmente la configuración es un poco más compleja, porque el servidor no sólo presenta archivos estáticos al cliente, sino que además:

1. Procesa un formulario estándar html y configura las reglas de filtrado del firewall de acuerdo al mismo.

2. Exporta estadísticas en formato XML[1], que pueden ser utilizadas por la aplicación "Flash" residente en el servidor, o cualquier otra aplicación con soporte XML

3. Envía al navegador la aplicación tipo "Flash"[2] (no confundir con la memoria Flash) nombrada anteriormente, para presentar las estadísticas, como se ve en la figura 8.2.

8.2. Estructura del servidor

La estructura del proceso servidor es muy sencilla comparada con servidores Web reales, debido a las limitaciones del hardware y del software. Como puede verse en la figura 8.1, el proceso atiende sólo una llamada y se reinicia, siguiendo la filosofía de

[1] XML es un estándar de intercambio de información desarrollado por el W3C

[2] ShockWave Flash es una tecnología creada por la empresa Macromedia, y es un estándar abierto utilizado en internet hace años.

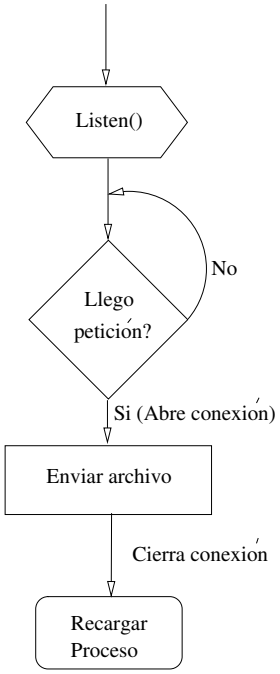


Figura 8.1.: Esquema del Proceso Servidor

los servidores reales, que reinician cada cierto tiempo sus procesos y así aumentan su confiabilidad, al eliminarse automáticamente cualquier bloqueo o mal funcionamiento que pudo haberse producido anteriormente.

8.3. Archivos

Existen sólo 6 archivos disponibles en el servidor, algunos con características especiales:

/index.html: Este archivo contiene la página principal del firewall, que está formado por una pequeña tabla con links al resto del sitio de configuración.

/frames.html: Es un archivo HTML estándar y estático que contiene una estructura especial denominada en la jerga html “Frameset”, cuyo objetivo es dividir el documento presentado en dos secciones independientes. Una estará dedicada a la presentación gráfica de estadísticas, mientras que la otra se dedicará a la configuración del dispositivo (puede apreciarse en la figura 8.2). El tamaño es muy pequeño, aproximadamente un centenar de caracteres, lo suficiente para incluirlo como una constante de C.

/Flash.html: Este archivo también es estático y contiene el código mínimo e indispensable para cargar la aplicación shockwave-Flash que está encargada de realizar la presentación final de las estadísticas. El código en sí se forma de los encabezados HTML mandatorios, más un par de “tags” especiales para embeber el código Flash en la página, utilizando el control Active-X de Flash versión 6. También posee un tamaño reducido y es almacenado como una constante de C.

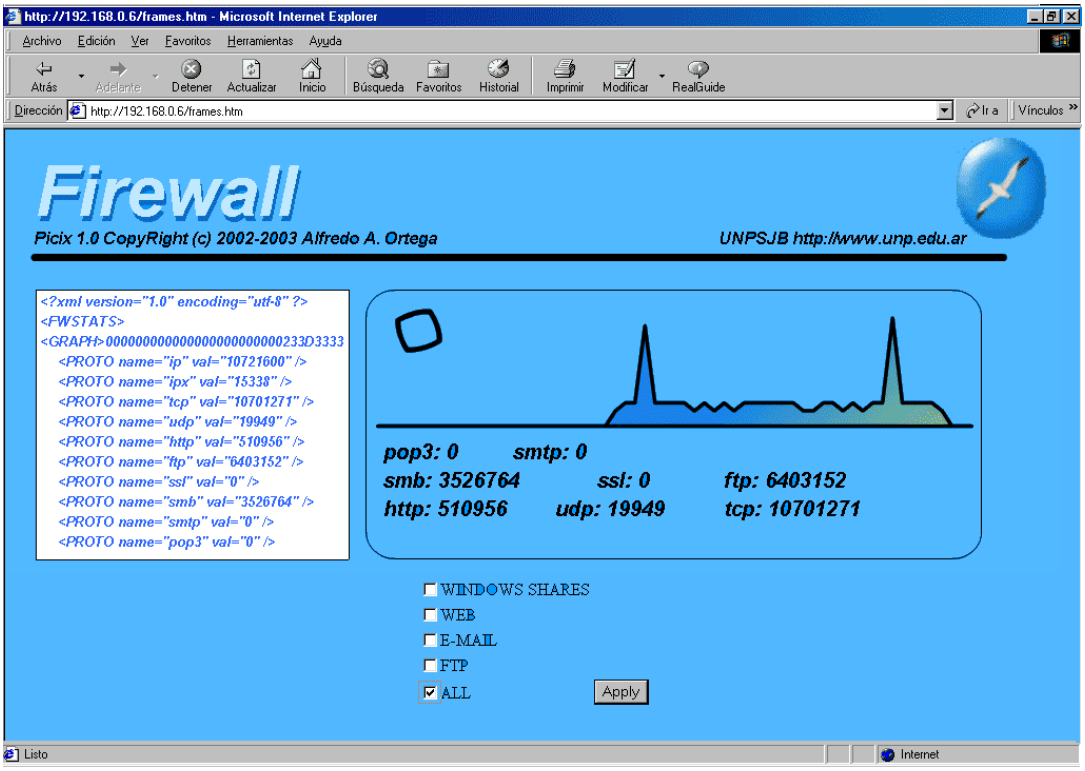


Figura 8.2.: Estadísticas y menú de configuración

/Stats.swf: Este es la aplicación Flash en sí. Es el archivo más grande que se transmite y consta de unos 3.5 Kbytes. Contiene el código necesario para cargar las estadísticas desde el dispositivo de firewall y presentarla realizando una gráfica del tráfico, y presentar numéricamente los contadores. El formato de este Archivo se trata con mayor profundidad en la sección 8.4.

/Menu.html: Este documento presenta el menú de configuración del dispositivo y aunque es parte del "frameset" que se carga en la página principal, puede ser cargado separadamente del resto si no se desean las estadísticas, que ocupan el reducido ancho de banda del dispositivo. Este documento difiere radicalmente de los demás ya que es generado dinámicamente de acuerdo a las reglas existentes en la memoria del firewall. Consta principalmente de un formulario estándar HTML, que permite seleccionar los servicios que el firewall dejará pasar. Cada servicio se traduce como una regla que es insertada dentro del firewall, por ejemplo, si se permite el servicio FTP, se agregará una regla aceptando paquetes TCP con el rango de puertos 20-21. Aunque el mecanismo es algo más complicado, ya que en este caso también hay que permitir el tráfico DNS y ARP, con lo que se generan combinaciones complejas.

/data.xml: La manera de exportar los datos estadísticos que recolecta el firewall, es mediante este archivo, generado dinámicamente cada vez que se lo pide (ver figura 8.3). El formato XML fue seleccionado por las siguientes características:

- Ser un estándar de intercambio abierto, permitiendo su utilización en multitud de aplicaciones

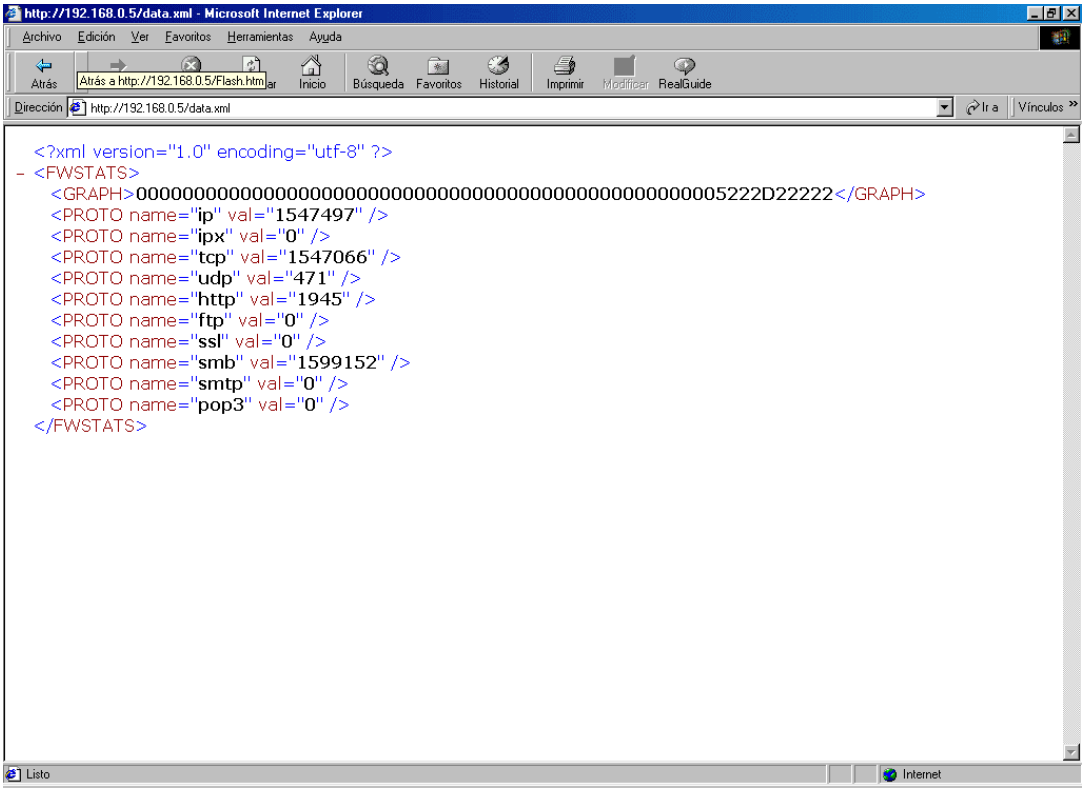


Figura 8.3.: Archivo XML de estadísticas visto desde Internet Explorer

- El lenguaje Flash (Action-Script, similar a Java-Script) lo soporta nativamente en su versión 6, por lo que el código necesario para procesar archivos de este tipo es confiable y compacto.
- Casi no hay diferencia en el tamaño y complejidad del código para generar un archivo de texto con formato propio que el necesario para generar un archivo XML[3].

Se almacena información adicional de los archivos en un array que contiene, para cada archivo, su nombre y los encabezados del protocolo HTTP 1.0[4]. Dentro de estos encabezados se transmite información muy importante, tal como el tipo MIME, que informa al navegador del tipo de archivo que se le está enviando (Los navegadores no lo reconocen por la extensión de los archivos, tal como hacen los sistemas operativos), y principalmente, información acerca de si un archivo que se está enviando se puede almacenar en la caché del navegador o no.

La memoria caché o memoria intermedia, se utiliza en los navegadores para mantener una copia local de los archivos descargados, y no tener que descargarlos nuevamente si no han sufrido ningún cambio. Con esto se acelera mucho la transferencia de páginas, al

[3] Aunque el código para aceptar y validar un archivo XML sea bastante complejo, de eso se encarga el intérprete de Flash.

[4] No confundir con HTML! éste es el *lenguaje* de las Páginas. HTTP es el *Protocolo* por el que se transmiten.

descargar sólo los archivos modificados. HTTP provee varios métodos, el más utilizado es el comando *"modified: "* que informa de la fecha de modificación del archivo, y permite al navegador decidir si se descarga o no. Pero como este dispositivo no posee un reloj interno capaz de almacenar fechas, se optó por otro comando, el denominado *"expire"* que permite especificar un tiempo de expiración de la página, combinado con otro comando, *"pragma: no-cache"* que indica directamente al navegador que no incluya el archivo en su memoria intermedia, y que cada vez que sea solicitado sea descargado. Esto es vital en el caso del archivo "data.xml" ya que si se almacenara en la memoria cache, nunca se vería la evolución del tráfico en la red, sino sólo la primer versión que fue descargada.

8.4. Software Flash Vs Java

Se consideró la utilización de un applet de Java en lugar del software Flash para la visualización de las estadísticas, ya que comparten muchas de las características más importantes, como lo compacto del código generado y amplias capacidades de comunicaciones pero para el caso particular de esta aplicación, el software Flash supera a Java en muchos aspectos, entre ellos:

1. El intérprete del software Flash ocupa menos de 1 megabyte, y está totalmente realizado en código nativo, por lo que su velocidad de carga en el navegador es despreciable. Por el contrario, el intérprete de Java ocupa unos 10 Megabytes y tarda varios segundos en cargar, aún en equipos potentes[5].

2. La capacidad gráfica del software Flash supera ampliamente a Java, ya que Flash puede verse como un sistema de animación con capacidades avanzadas de programación, mientras que Java es un sistema de máquina virtual, que no fue diseñado específicamente para realizar animaciones y presentaciones gráficas.

La mayoría de los sistemas conectados a Internet disponen del intérprete del software Flash instalado dentro de su navegador, ya que hoy en día es la tecnología preferida en muchos sitios populares, aunque por ser la versión utilizada relativamente reciente (Flash 6) , quizás haya que actualizar el intérprete.

8.5. Action-Script

Quizás sea de interés una descripción del lenguaje que Flash permite utilizar dentro de sus animaciones.

Action-Script es muy similar en su sintaxis y utilización a JavaScript[6] , otro lenguaje de scripting utilizado para crear páginas dinámicas. Está orientado a objetos y en su versión 6.0, ha evolucionado hasta convertirse en un avanzado lenguaje orientado a objetos,

[5]Específicamente en entornos Windows, Java tiene un pobre rendimiento como applet, quizás como resultado de la estrategia de Microsoft que ignorando a la tecnología Java, incluye un intérprete defectuoso y obsoleto con su sistema operativo. La implementación actual de Sun Microsystems tampoco sobresale por su velocidad en equipos modestos.

[6]Que a su vez, es muy similar a Java o C++ en su sintaxis.

capaz de efectuar cálculos muy complejos y ocupar un mínimo espacio de memoria y código. Aunque la estructura de un archivo Shockwave/Flash estaba orientada a “cuadros” y “capas” de animación, en las últimas versiones está cambiando a un paradigma de objetos e inter-relaciones. En particular, posee herramientas para manejar conexiones TCP mediante archivos de texto y un soporte robusto de XML[7]. Aunque Flash es una aplicación comercial, su formato de archivo es abierto, y la especificación puede descargarse desde el sitio comercial del su creador, Macromedia http://www.macromedia.com

8.6. Eficiencia

La eficiencia del servidor Web depende primordialmente de la eficiencia de la pila TCP/IP subyacente, aunque de por sí, el sistema no admite comunicaciones simultáneas, por lo que en éste sentido puede verse una pérdida de eficiencia relacionada con el hecho de que los navegadores suelen pedir varios archivos a la vez, y al no poder descargarlos, se producen retardos y errores en la descarga. De todos modos, aunque la conexión tiene un máximo teórico de 10 MegaBits por segundo, puede esperarse una cifra bastante menor, debido a que la pila TCP/IP, al tener que funcionar con muy poca memoria no dispone de buffers de tamaño suficiente, y los segmentos TCP transportan muy pocos datos[8], reduciendo la perfomance.

8.7. Seguridad

En un sistema de estas características, la seguridad es vital, y permitir la administración por vía Web, introduce una seria amenaza, que se produce al poder modificar o anular completamente al sistema de firewall desde cualquier lugar del mundo, ya que justamente esa funcionalidad es la que permite el conjunto de protocolos TCP/IP. Se necesita un método de proveer seguridad durante la configuración del sistema de firewall. A continuación se nombrarán los métodos propuestos:

8.7.1. Anulación completa de la pila TCP/IP

En situaciones de alto riesgo, se podría directamente desactivar el subsistema TCP/IP y administrar el dispositivo utilizando la consola serial. Esto anularía el método secundario de configuración (interfaz Web) pero el firewall podría seguir configurándose mediante la línea de comandos, opción mucho más segura ya que se necesita acceso físico al dispositivo para realizar las modificaciones.

[7] Esta pequeña aplicación es capaz de extraer mucha información de un archivo XML dañado, mientras que, por ejemplo, Internet Explorer se niega a procesarlo.

[8] En la implementación actual, unos 70 Bytes por segmento TCP, que por supuesto necesita de un paquete Ethernet completo para su transmisión.

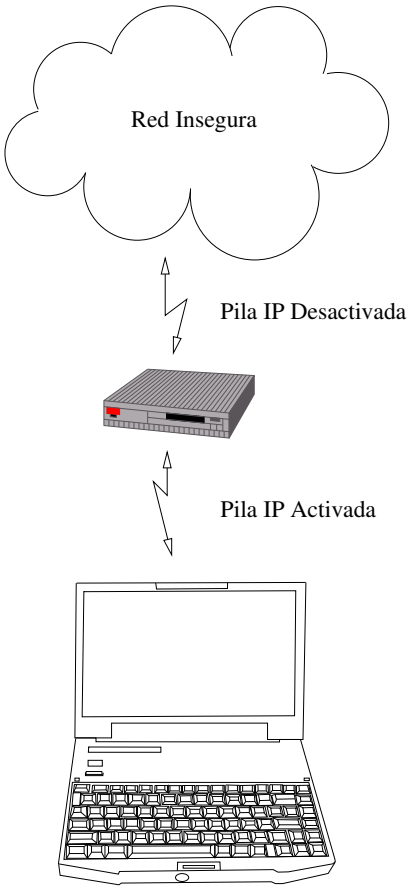


Figura 8.4.: Modelo de seguridad adoptado

8.7.2. Métodos de autenticación

La desactivación completa de la interfaz Web es una opción un poco drástica, por lo que se pensó implementar algún método para autenticar al usuario del sistema.

La utilización de palabras clave como autenticación para la interfaz de administración aumenta un poco la seguridad, pero hay que tener en cuenta que el microcontrolador no posee potencia suficiente como para realizar una comunicación segura, utilizando por ejemplo, la tecnología SSL. Transmitir un password sin cifrar a través de la red ya no es una opción considerada segura en ningún sistema moderno.

8.7.3. Anulación parcial de la pila TCP/IP

Este método, que fue el utilizado finalmente, no involucra el servidor Web, sino la modificación de la pila TCP/IP para que acepte conexiones sólo de una interfaz Ethernet, que se debería conectar a la sección “interna” de la red a proteger, como se ilustra en la figura 8.4. De este modo la red exterior estaría imposibilitada de acceder al dispositivo ya que directamente éste no respondería a ningún pedido por esa interfaz. Esto se maneja mediante el comando `ip`, tal como se muestra en la sección 7.6 en la página 74.

Nota: No confundir este método con el generalmente utilizado en firewalls normales, que filtran el acceso de acuerdo con la dirección de red. En este caso no se considera

ninguna dirección, sólo se responde a conexiones provenientes de una de las interfaces. Invirtiendo los cables de red se habilitaría el acceso a la otra red, pero nunca se podría acceder de ambos lados del firewall.

8.8. Codificación gzip

El protocolo HTTP permite que todo archivo transmitido posea una codificación adicional, y lo hace mediante el campo "`content- enconding`" de una respuesta HTTP. Una codificación disponible es el algoritmo *gzip*, que es un conocido y eficiente método de compresión de datos. El navegador deberá en este caso descomprimir automáticamente todo el contenido que utiliza esta opción (Internet Explorer 5 posee soporte para descompresión automática de la codificación gzip). Esto no sólo ahorraría el preciado espacio en la memoria del microcontrolador, sino que aceleraría las transferencias, ya que el contenido se transmite comprimido.

Se realizaron numerosas pruebas con esta opción, que ahorra aproximadamente 500 bytes de memoria Flash y reduce de manera apreciable el tiempo de transferencia de las páginas, pero surgieron varios problemas por parte de Internet Explorer 5, que al utilizar esta técnica sufre reiterados bloqueos y mal funcionamiento. Estos "bugs" son conocidos y en teoría, fueron solucionados en la versión 6 de Internet Explorer (no fue probado).

Para la presentación final se decidió desechar esta opción, ya que se priorizó la estabilidad del sistema por sobre las ventajas que ofrece la codificación gzip.

9. Firewall

9.1. Clasificación

Luego de la descripción de los componentes que hacen al sistema, se detallará el corazón del mismo, ahora muy simplificado debido a todo el software de soporte que se ha realizado.

Se realizará una rápida clasificación para ubicar este trabajo:

Firewalls "Filtro" sin inspección: En este tipo encuadra el sistema desarrollado, y es uno de los tipos de cortafuegos más sencillos de realizar, ya que no requieren mucha memoria ni procesamiento. Filtran los paquetes que lo atraviesan en base a unas reglas que define el usuario.

Firewalls "Filtro" con inspección: Muy parecido al tipo superior, pero mantiene un control adicional sobre la estructura y orden de los paquetes que lo atraviesan, pudiéndose especificar reglas más detalladas, tales como filtrar paquetes que no pertenezcan a una conexión TCP pre-establecida, o simplemente eliminar todo tráfico que el sistema considere mal intencionado. El dispositivo desarrollado podría cumplir este tipo de función si se le adicionara algunos kilobytes de memoria RAM.

Firewalls de enmascaramiento: También denominadas NAT (Network Address Translators, Traductores de dirección de red), ocultan una sub-red traduciendo sus direcciones, generalmente privadas, a direcciones públicas. Son uno de los tipos de firewall más seguros.

Proxys: Inspeccionan el tráfico en la capa de aplicación, a diferencia de los tipos nombrados anteriormente, que trabajan específicamente en las capas de red y transporte, como se ve en la figura 9.1. Se requiere un poder de proceso superior que en los otros tipos.

El firewall desarrollado es de tipo "filtro" sin inspección, debido a la poca memoria del dispositivo (1.5 Kb). Teniendo en cuenta que se necesita un promedio de 25 bytes para mantener la información acerca de cada conexión para los demás tipos, y tomando un máximo de 100 conexiones TCP simultaneas, se necesitaría al menos $100 * 25 bytes = 2{,}5 Kb$ de memoria adicional para obtener un dispositivo usable. El poder de cálculo necesario no es tan elevado, excepto en el caso de los proxys, por lo que utilizando una memoria externa y quizás un microcontrolador con mayor cantidad de memoria flash, podrían implementarse los demás tipos de manera muy eficiente.

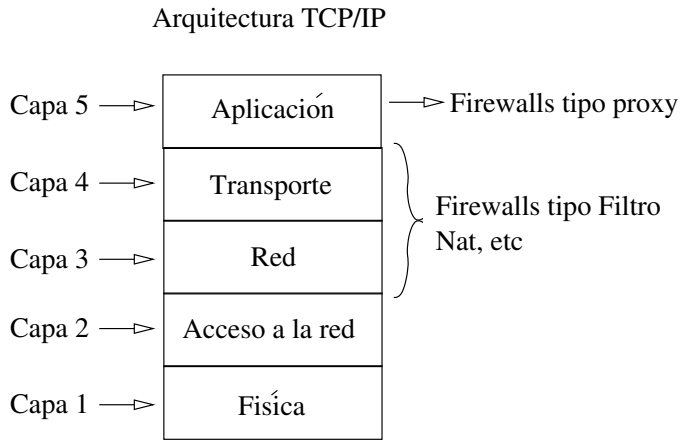


Figura 9.1.: Ubicacíon de los tipos de firewall en las capas de red TCP/IP

9.2. Estructura general

Un firewall tipo filtro tiene una estructura muy sencilla, ya que después de todo, sólo se necesita una función que decida si deja pasar el paquete o no. Esta función, se denomina `firewall()` en el sistema. Como puede verse en la figura 9.2, hay que analizar cada paquete entrante para recopilar estadísticas. Además, antes de efectuar la decisión de reenviar o no el paquete, debe ser procesado por la pila IP. Si no, se corre el riesgo realizar por error alguna regla demasiado restrictiva y bloquear todo acceso al sistema mediante la interfaz Web, debiendo reconfigurarlo mediante la línea de comandos.[1].

El proceso de firewall corre en un loop infinito, primero seleccionando una de las interfaces, analizando el paquete que provenga de esa interfaz, y luego de calcular estadísticas, procesarlo mediante la pila IP y decidiendo si se retransmite o no, el loop recomienza, pero esta vez revisando la interfaz contraria, ya que los paquetes pueden atravesar en ambos sentidos el dispositivo.

9.3. Mecanismo de configuración

Como se nombró en el comienzo, la manera más intuitiva de configurar un firewall tipo filtro es mediante un conjunto de reglas que permitan especificar el tipo de paquetes que pueden atravesar, o deben bloquearse. Se adoptó un funcionamiento similar al comando "`ipchains`" o su versión actualizada "`iptables`" de Linux, obviamente con un conjunto de opciones simplificado. Como se ilustra en la figura 9.3, en Linux existen tres conjuntos de reglas principales de filtrado:

1. El grupo "INPUT" que se aplica a los paquetes cuyo destino es el propio host (computadora).

[1]Esto suele pasar en los sistemas de Firewall comunes, ya que proveen una flexibilidad superior a la nuestra, y con la flexibilidad se producen este tipo de problemas.

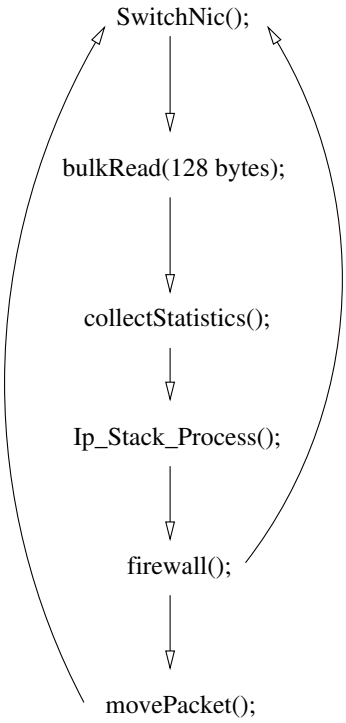


Figura 9.2.: Diagrama de pseudocódigo del firewall implementado

2. El grupo “OUTPUT” que se aplica a los paquetes que salen desde el host hacia la red.

3. El grupo “FORWARD” que se aplica a los paquetes que llegan al host con destino a otro, o sea, están en tránsito.

Estos grupos se denominan “cadenas” de reglas, y en Linux ahora se pueden agrupar en *tablas*, de ahí surgen los nombre “`ipChains`” e “`ipTables`”.

Estos dos comandos son interfaces para el sistema de firewall de Linux, denominado *netFilter*. Toda esta explicación viene al caso debido a que nuestro funciona de la misma manera, sólo que posee únicamente la cadena “FORWARD”, o sea, no impone ninguna restricción sobre los paquetes que entran o salen del mismo firewall, ya que dichos paquetes sólo obedecerían al mecanismo de configuración Web, y en todos los casos, no es posible efectuar ningún ataque sobre el dispositivo de ésta manera, como se explica en la sección 8.7 en la página 81.

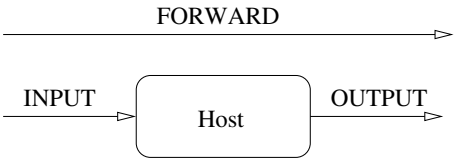


Figura 9.3.: “Cadenas” del Firewall *netFilter* del sistema Linux.

9.4. Reglas

Las reglas deben contener la información necesaria para identificar un paquete y efectuar una decisión sobre el mismo. Se definieron dos tipos de reglas: una regla “completa”

Listado 9.1: Estructura fwRule

```
typedef struct
{
ipAddr ipIn;
ipAddr ipOut;
unsigned char proto;
portRange portIn;
portRange portOut;
unsigned char dest;
unsigned long bytes;
} fwRule;
```

y una regla “corta”. Se dirá de antemano que en el desarrollo final se utilizaron exclusivamente las reglas cortas, por no resultar suficiente la memoria para implementar totalmente las reglas “completas”.

En la estructura de la regla “completa” denominada `fwRule` (listado 9.1) puede apreciarse que se almacena información suficiente para identificar un paquete por su dirección de origen o destino, o su puerto de origen y destino. El campo `bytes` es un contador de la cantidad de bytes de los paquetes que coinciden con esta regla, y el campo `dest` especifica el destino del paquete, que puede ser:

POLICY_ACCEPT: El paquete se acepta y se envía inmediatamente a la otra interfaz.

POLICY_DISCARD: El paquete se rechaza y es descartado.

POLICY_REJECT: También se rechaza el paquete, pero se envía un paquete ICMP informando de dicha acción[2].

RULE_EMPTY: Especifica que la regla está vacía o fue eliminada, y no debe tomarse en cuenta. Se pasa a la próxima regla en la lista, o si se llegó al final de la misma, el destino del paquete lo decide la política final del sistema.

RULE_GRAPH: El largo del paquete (en bytes) es agregado a la suma total del gráfico de tráfico (estadísticas), pero no se toma ninguna decisión sobre el mismo, sino que se pasa a la próxima regla.

Teniendo en cuenta que las reglas deben almacenarse en la memoria EEPROM, y que sólo se disponen de 256 bytes de la misma, sólo se podrían almacenar 7 u 8 de estas

[2]Este tipo de destino no fue implementado en la versión final.

Listado 9.2: Estructura fwRuleShort

```
typedef struct
{
portRange pr;
unsigned char dest;
unsigned long bytes;
} fwRuleShort;
```

reglas (El largo de cada regla completa es de 24 bytes). Por este motivo, el sistema trabaja con éste otro tipo de de regla, conocida como "Regla corta" cuya estructura se denomina `fwRuleShort` (listado 9.2). Esta régla no permite discriminar paquetes por la dirección IP, sino sólo por rango de puertos.

Cada regla ocupa sólo 9 bytes, y es posible almacenar más de 20 en la memoria EEPROM. En la versión final del software sólo se implementaron estas últimas, aunque toda la estructura para la implementación de las reglas completas está codificada y sólo hace falta soporte por parte de la línea de comandos.

9.5. Estadísticas

Cada paquete se procesa por la función `collectStatistics()`, que de hecho sólo inspecciona los encabezados IP de los mismos, para extraer información acerca del tamaño del "*payload*" (cargamento, o datos útiles) de los paquetes, ya que todas las estadísticas se limitan a calcular el tráfico que atraviesa el firewall, en bytes. También se realiza una inspección sobre paquetes IPX, pero no se registra mayor información sobre los mismos que la suma total del todo el tráfico de este protocolo. Como contrapartida, el tráfico IP es desglosado en sus diferentes capas y protocolos.

También se registra un gráfico, que se forma por la sumatoria del tráfico que capturan todas las reglas con destino seteado en "RULE_GRAPH", por lo que pueden setearse reglas para graficar el tráfico IP, o específicas para graficar la actividad Web o FTP.

Cada regla a su vez, mantiene un contador de la cantidad de bytes que contienen los paquetes que fueron capturados por las mismas. Todos los contadores de las estadísticas tienen un ancho de 32 bits, por lo que pueden registrar un máximo de 4 Gigabytes, aproximadamente.

Las estadísticas pueden obtenerse por vía Web, como se ilustra en la figura 8.2 en la página 78, o mediante el comando "`ipstats`", como se ve en la figura 9.4.

Los protocolos listados son:

- Capa 3: IPX, IP, otros.
- Capa 4: TCP,UDP, otros.

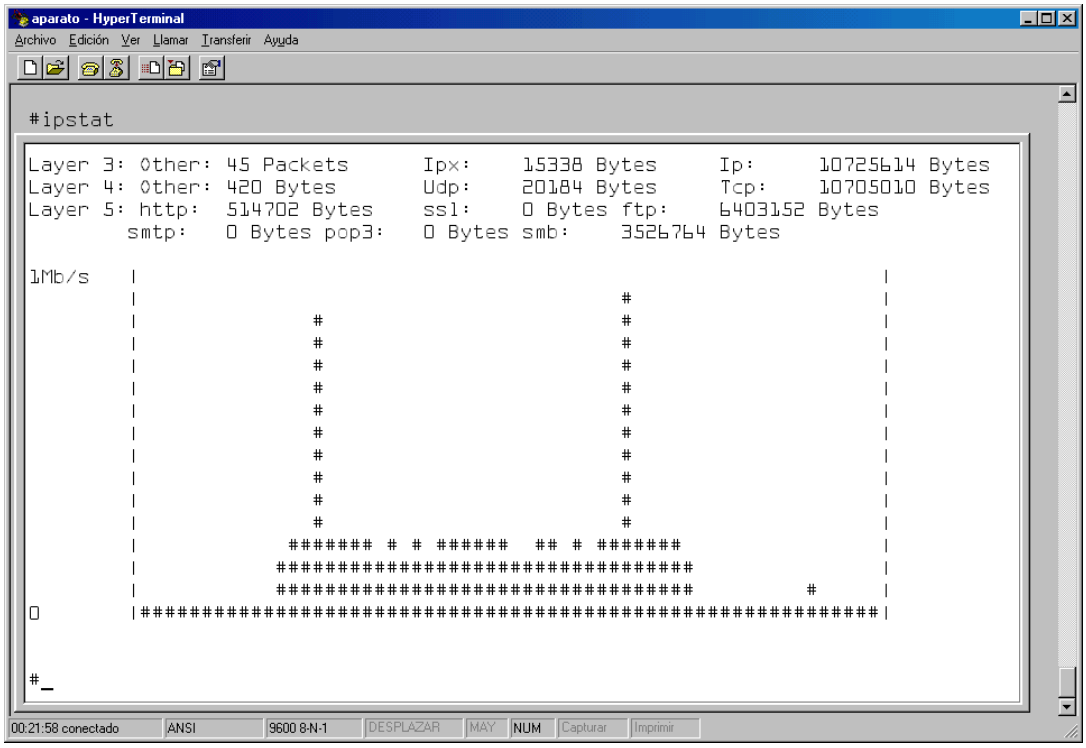


Figura 9.4.: Estadísticas con el comando “`ipstats`”

- Capa 5: http, ssl, ftp, smtp, smb
- Por defecto, la gráfica mide el tráfico total que atraviesa el firewall, pero esto puede modificarse mediante reglas especiales.

9.6. Procesamiento de la Pila TCP/IP

La pila TCP/IP siempre procesa los paquetes, sean rechazados o no. Esto es debido mayormente al protocolo ARP, que como dicta el estándar, debe realizar el mantenimiento de su tabla con todos los paquetes. Esto es un poco engorroso en nuestro caso ya que la interfaz Ethernet trabaja en modo promiscuo y por lo tanto recibe y procesa todos los paquetes.

La pila IP trabaja sólo con los primeros 128 bytes del paquete, principalmente porque éste es el tamaño máximo de su buffer. Paquetes mayores son procesados sólo parcialmente.

Un caso especial son los paquetes “Broadcast” ya que no solamente deben ser procesados por la pila, sino que también deben ser transmitidos en su totalidad. Surge un conflicto, ya que la pila IP sólo procesa los primeros 128 bytes, mientras que un paquete broadcast puede y suele superar este largo. El comportamiento adoptado es el de procesar sólo los paquetes de broadcast de tipo “ARP” que siempre serán inferiores a 128 bytes, y los demás se envían a la otra interfaz sin modificar.

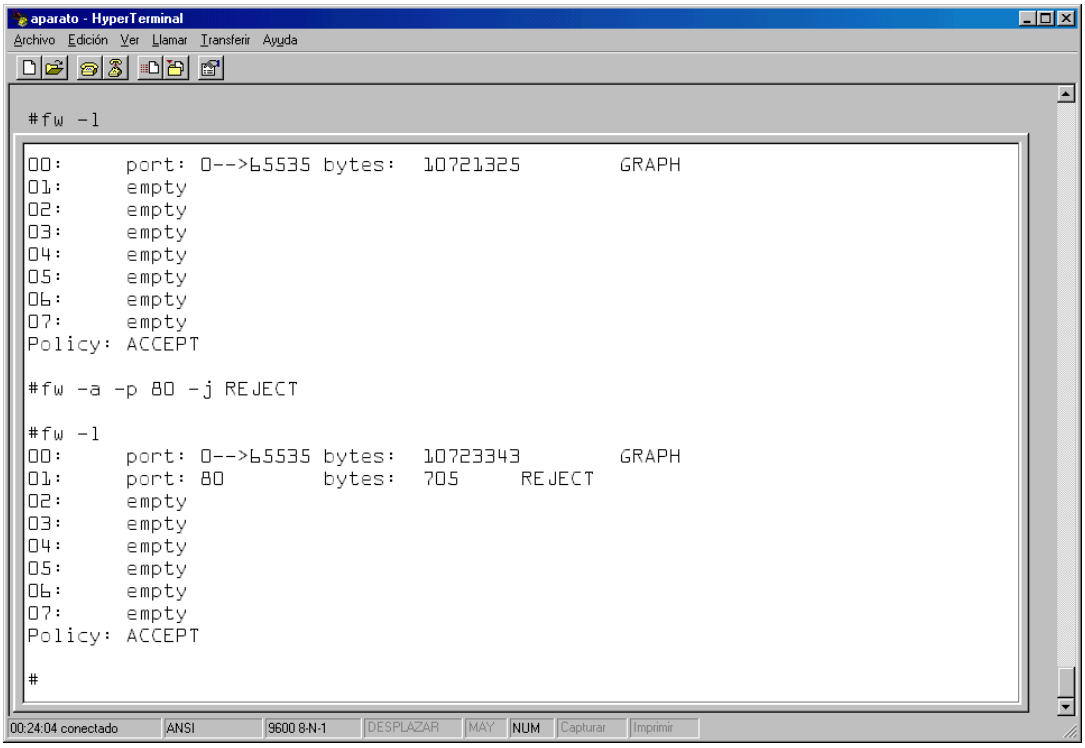


Figura 9.5.: Configuración mediante el comando "`fw`"

9.7. Línea de Comando

El Firewall se configura mediante un sólo comando, denominado "`fw`" con multitud de opciones, entre ellas:

-l Muestra todas las reglas y contadores

-c [regla]: Elimina la regla especificada

-a Agrega una regla al final de la lista

-i Inserta una regla al principio de la lista

-p p1-p2: Especifica un rango de puertos

-j destino: Especifica el destino de la regla.

Además de poder ver las estadísticas mediante la interfaz Web, es posible acceder a una versión de texto de las mismas mediante el comando "`ipstats`" detallado en la sección 9.5, que sólo tiene dos modificadores:

-c para inicializar todas las estadísticas y

-a para realizar una pequeña animación de la gráfica, de 10 segundos de duración.

9.8. Ejemplos

Para rechazar todo paquete utilizado por netBIOS (Puertos 137,138 y 139) sobre TCP/IP:

```
fw - a - p 137 - 139 - j REJECT
```

Para aceptar el tráfico HTTP:

```
fw - a - p 80 - j ACCEPT
```

Para que el firewall rechace todos los paquetes que no coincidan con ninguna regla:

```
fw - p REJECT
```

Para que el tráfico del protocolo FTP pueda apreciarse como una gráfica:

```
fw - a - p 20 - 21 - j GRAPH
```

En la figura 9.5 se muestra el uso del comando "`fw`" en el sistema.

9.9. Análisis de red

Para finalizar, se realizó un estudio del impacto que causa el firewall en una red. Para evitar cualquier modificación por parte de agentes externos, las pruebas se realizarán en una red cerrada y controlada, formada sólo por dos nodos conectados mediante una red full-duplex de 10 Megabits. Se tomaron medidas de RTT (Round Trip Time) y flujo de datos primeramente realizando una conexión directa entre ambos nodos utilizando un cable cross-over, y luego instalando el Firewall entre ambos sin ninguna restricción.

El protocolo elegido fue netBIOS sobre TCP/IP y la prueba fue el copiado de un archivo entre dos directorios compartidos de Windows, operación relativamente común en un ambiente real. Hay que tener en cuenta que la velocidad alcanzada en esta prueba refleja valores máximos.

En la figura 9.6 se aprecian los datos acerca de las mediciones, de los cuales el más significativo es el denominado "`Avg. Mbit/sec`" que muestra el promedio del tráfico. Se aprecia una disminución significativa en el tráfico, de 5 Mbits/seg hasta 1.8 Mbits/seg, causado por el procesamiento adicional introducido por el firewall. No se alcanzó el límite teórico de 10 Mbits/s ya que al no tratarse de una prueba totalmente "pura", se producen retardos tales como el disco rígido, el procesamiento propio de las PCs, etc.

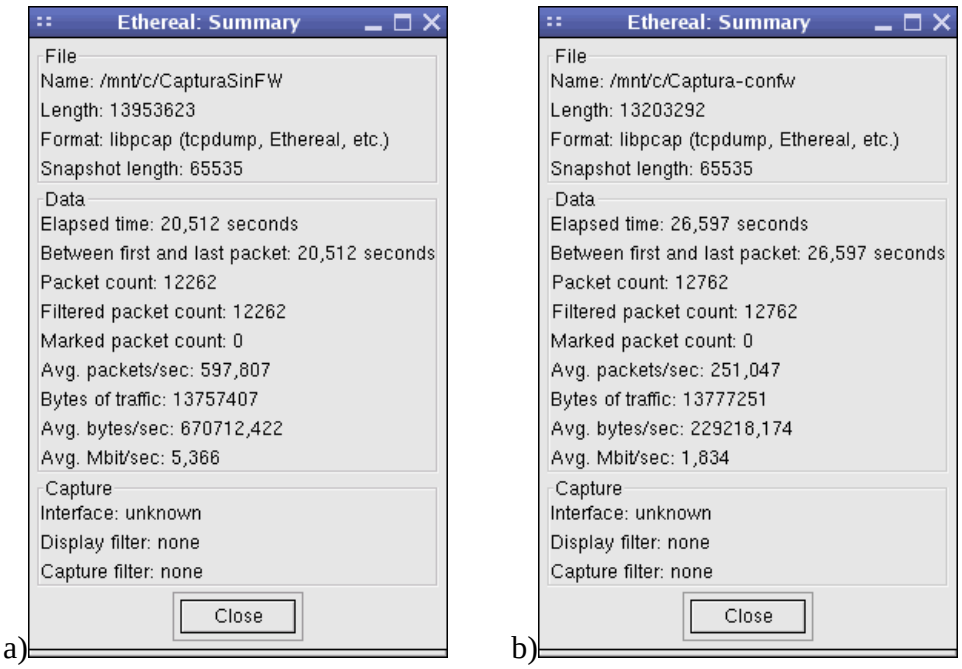


Figura 9.6.: Resumen de ambos análisis: a) conexión directa b) conexión con Firewall

A continuación se presentan dos análisis realizados mediante la herramienta `Ethereal` de análisis de tráfico:

1. Análisis de RTT (Round Trip Time, Tiempo de viaje redondo) que apunta a medir el retardo introducido por el sistema de firewall.

2. Análisis de Flujo, mide la modificación en la cantidad máxima de información por unidad de tiempo que es posible transmitir al utilizar el sistema de firewall.

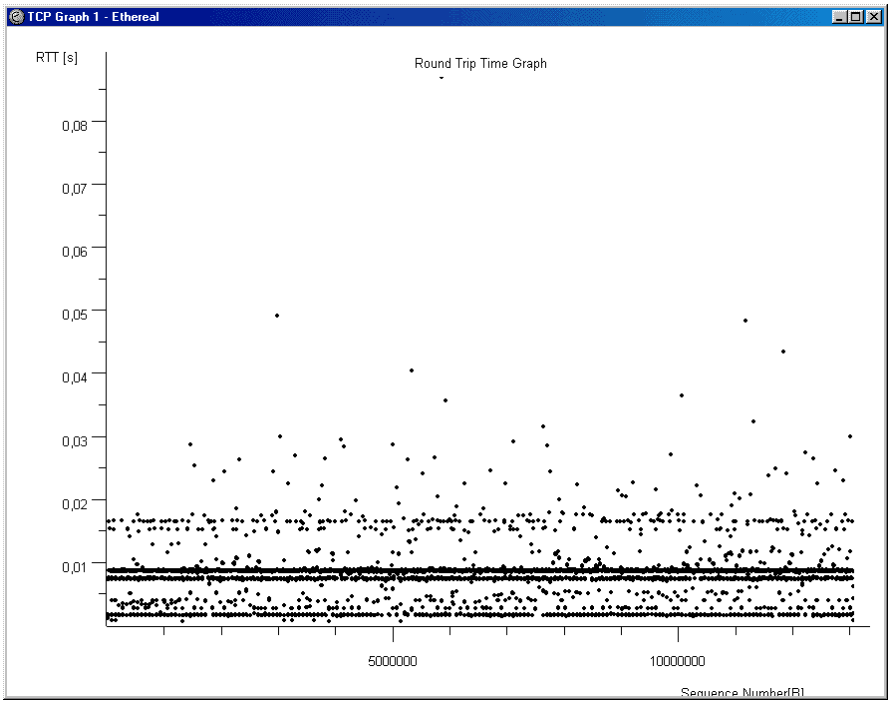


Figura 9.7.: Análisis RTT, conexión directa

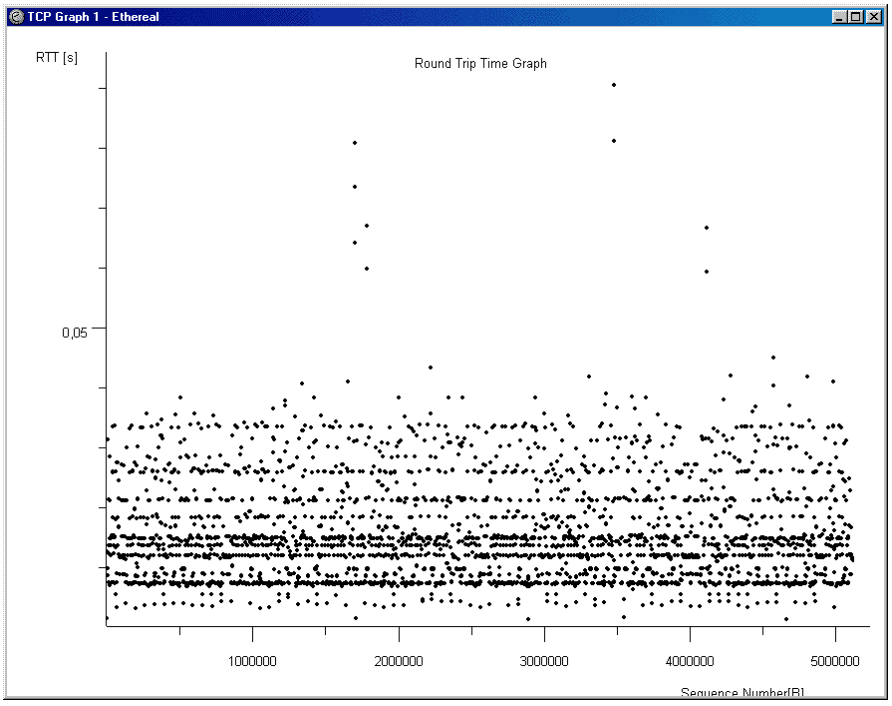


Figura 9.8.: Análisis RTT, con Firewall

Se aprecia un aumento del RTT que se produce por el tiempo de procesamiento adicional que introduce el Firewall.

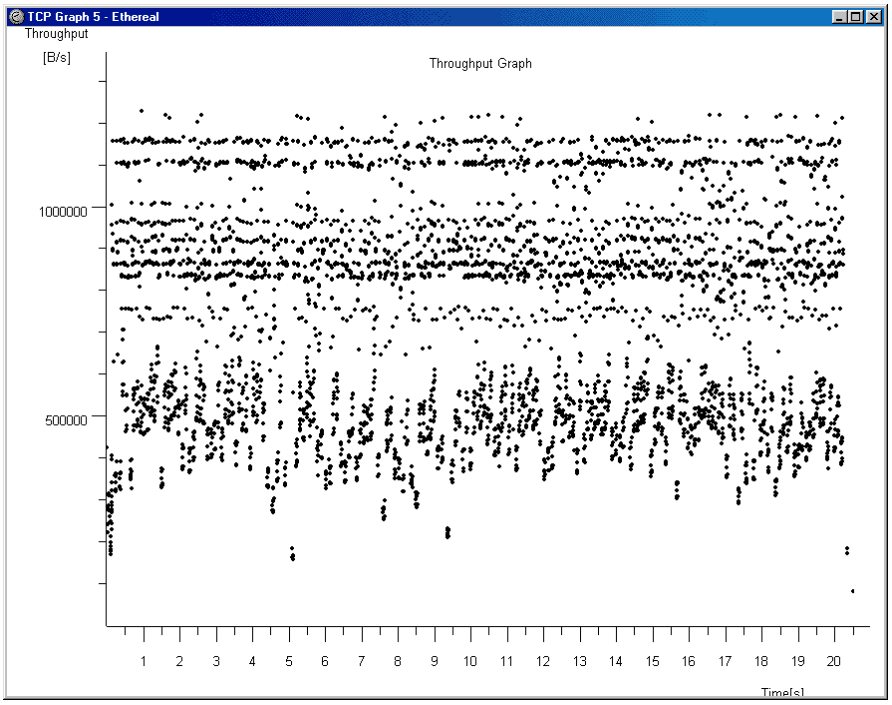


Figura 9.9.: Flujo (Throughput), conexión directa

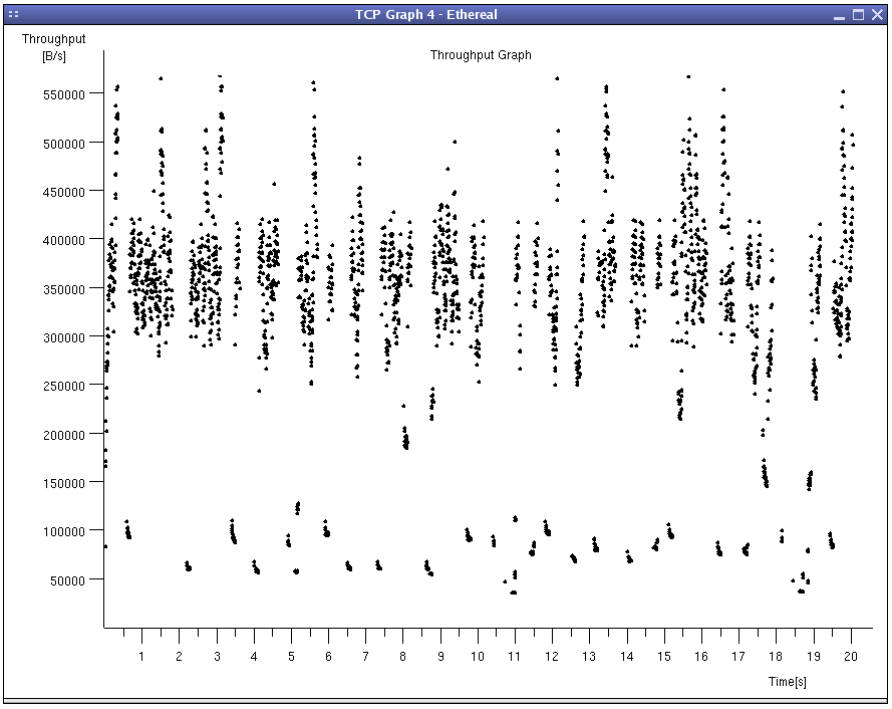


Figura 9.10.: Flujo (Throughput), con Firewall

Se aprecia un patrón similar en la transmisión, aunque a escalas diferentes, ya que el flujo de datos con el Firewall es menor.

10. Conclusiones

10.1. Resumen

- Se lograron todos los objetivos del proyecto, implementando un sistema de firewall portatil y de fácil configuración utilizando el Microcontrolador propuesto. Dichos objetivos se alcanzaron dentro del tiempo previsto para la finalización de la tesina.
- El Microcontrolador PIC-18 resultó tener una gran potencia, a pesar de sus limitados 10 MIPS (limitados en apariencia). El cuello de botella en este caso estuvo en realidad en el diseño del bus, que posee un máximo de 2 Megabit/segundo. El dispositivo tiene suficiente velocidad para filtrar redes WAN, DSL, inalámbricas o hasta redes Ethernet a 10 Megabits, que en la práctica nunca alcanzan su máxima capacidad teórica.
- Al utilizar un sistema operativo se simplifica en gran medida el diseño del software, pero aumenta los errores producidos por paralelismo y bloqueo mutuo, que son muy difíciles de corregir. Se adoptaron mecanismos simples de prevención tales como semáforos tipo "`mutex`" para evitar bloqueos mutuos y corrupción de memoria.
- La parte más compleja del sistema es la pila IP, que aunque no sea el objetivo principal del sistema, aumenta mucho el valor total del diseño, ya que otros proyectos basados en el mismo podrían hacer de éste su componente principal de comunicaciones.

10.2. Contenido de la carrera en relación al proyecto

Las cátedras que se dictan durante la carrera de Licenciatura en Informática en la Universidad Nacional de la Patagonia San Juan Bosco cubren casi en su totalidad los temas necesarios para concretar un proyecto de estas características, con la excepción de los apartados de hardware y microcontroladores, que de ser considerados de mayor importancia en el futuro podrían llegar realizarse cursos adicionales sobre el tema.

10.3. Experiencia adquirida

1. Investigación sobre el medio de Internet, y sobre manuales técnicos de componentes electrónicos.
2. Aplicación de los contenidos de la carrera.

3. Toma de decisiones acerca de tecnologías y métodos a seguir durante un proyecto a mediano plazo.
4. Desarrollo de software de base y de bajo nivel.
5. Desarrollo de software de comunicaciones a niveles bajo e intermedio.
6. Experiencia en otras arquitecturas y especialmente en la nueva generación de microcontroladores.
7. Desarrollo de documentación y divulgación de investigaciones.

10.4. Notas finales

El desarrollo fue finalmente completado, y como resultado se obtuvo una plataforma con muchas posibilidades, además de actuar como “Firewall”, ya que un dispositivo de bajo precio y alta capacidad de cálculo como este tiene múltiples usos. Como un ejemplo, se podrían utilizar varios periféricos que quedaron sin utilizarse en este proyecto, tales como las entradas y salidas analógicas, múltiples puertos digitales bidireccionales, interfaces sincrónicas, etc.

El advenimiento de nuevas tecnologías de microcontroladores amplían en gran medida el campo del profesional en Ciencias de la Computación, hacia áreas que anteriormente estaban limitados a la Ingeniería Electrónica o ambientes industriales.

Bibliografía

[Pressman:1997] Roger S. Pressman (1997), **Ingeniería de Software, un enfoque práctico**, Cuarta Ed. McGraw-Hill

[Comer:1995] Douglas E. Comer (1995), **Redes Globales de Información con Internet y TCP/IP**, Principios básicos, protocolos y arquitectura, Tercera Ed., Prentice-Hall.

[Sebesta:1996] Robert W. Sebesta (1996), **Concepts of Programming Languages**, Third Ed. , Addison-Wesley.

[Stallings:2000] William Stallings (2000), **Comunicaciones y Redes de Computadores**, 6ªedición, Prentice-Hall.

[Tanenbaum:1992] Andrew S. Tanenbaum (1992), **Modern Operating systems,** Prentice-Hall

[UYM:2000] Usategui, Yesa y Martínez (2000), **Microcontroladores PIC**, Diseño práctico de aplicaciones, McGraw-Hill

A. Software Utilizado

MPLAB(R) IDE v6.12, Entorno integrado de desarrollo,

microchip http://www.microchip.com

MPLAB(R) C18 v2.10.06, Compilador para Procesadores 18Cxxx,

microchip http://www.microchip.com

IC-PROG v1.05, Multi-programador de Dispositivos Flash,

icprog http://www.ic-prog.com

Jolt PIC18F Bootloader, Software de Booteo para Procesadores 18Fxxx,

Jolt Bootloader
http://members.rogers.com/martin.dubuc/Bootloader

Ethereal v 0.9.6, Analizador de Protocolos de Red,

ethereal http://www.ethereal.com

Hyperterminal-win98 o Minicom-Linux, Emulador de Terminal.

Redhat Linux 8.0, Sistema Operativo,

redhat http://www.redhat.com

Windows 98, Sistema Operativo,

Microsoft http://www.microsoft.com

Salvo, Sistema Operativo

Pumpkin, Inc. http://www.pumpkininc.com

Nmap, Network Scanner (Escaner de red),

Nmap http://www.insecure.org/nmap

Latex/Lyx, Sistema de Composición de Documentos,

Latex/Lyx http://www.lyx.org

Xfig, Paquete de dibujo vectorial

Xfig http://www.xfig.org

MrProject, Manejador de proyectos

MrProject http://mrproject.codefactory.se

B. Glosario

10Base-T Nombre técnico para el "par" trenzado de Ethernet, a la velocidad de 10 Megabits

802.3 Estándar IEEE para Ethernet

802.11 Estándar IEEE para Redes Inalámbricas

ACK Abreviatura de *acknowledgement* (reconocimiento o acuse de recibo)

ANSI *(American National Standards Institute)* Grupo que define los estándares de Estados Unidos para la industria del procesamiento de información. ANSI participa en la definición de los estándares de protocolos de Red.

ARP *(Address Resolution Protocol)* Protocolo TCP/IP utilizado para asignar una dirección IP de alto nivel a una dirección de hardware físico de bajo nivel. ARP se utiliza a través de una sola red física y está limitada a redes que soportan difusión de hardware.

AUI Abreviatura de *Attachment Unit Interface*, el conector utilizado en las redes Ethernet de cable grueso.

baud Literalmente, el número de veces que una señal puede cambiar por segundo en una línea de transmisión.

bps (bits per second) Medida de la razón de trasmisión de datos.

broadcast (difusión) Sistema de entrega que proporciona la copia de un paquete dado a todos los anfitriones conectados para la difusión del paquete.

DNS *(Domain Name System)* Sistema de base de datos distribuida en línea y utilizado para transformar nombre de máquina en direcciones IP que puedan leer los usuarios.

EIA *(Electronics Industry Association)* Organización de estándares para la industria electrónica. Conocida por los estándares RS232C y RS422.

Ethernet Popular tecnología de red de área local inventada en el Palo Alto Research Center, de Xerox Corporation.

Firewall (Cortafuegos, muro de seguridad) Dispositivos colocados entre la organización interna de una red de redes y su conexión con redes de redes externas, con el fin de proporcionar seguridad.

FTP *(File Transfer Protocol)* Protocolo estándar de alto nivel del TCP/IP que sirve para transferir archivos de una máquina a otra. El FTP utiliza TCP.

host (Anfritrión) Cualquier sistema de computadora de usuario final que se conecta a una red. Los anfitriones abarcan desde computadoras personales hasta supercomputadoras.

hub (Concentrador) Dispositivo electrónico al que se conectan varias computadoras, por lo general mediante un cable de par trenzado.

ICMP *(Internet Control Message Protocol)* Parte integral del protocolo de Internet (IP) que resuelve errores y controla los mensajes. ICMP incluye una solicitud/réplica de eco utilizada para probar si un destino es accesible y responde.

Internet Conjunto de redes y ruteadores que abarca multitud de países y utiliza los protocolos TCP/IP para formar una sola red virtual cooperativa.

IP *(Internet Protocol)* Protocolo estándar que define a los datagramas IP como la unidad de información que pasa a través de una red de redes y proporciona las bases para el servicio de entrega de paquetes sin conexión y con el mejor esfuerzo.

ISR (Interrupt Service Routine) Rutina de servicio de interrupción, es llamada cada vez que se produce una interrupción de hardware.

LAN *(Local Area Network)* Cualquier tecnología de red física diseñada para cubrir distancias cortas.

MAC *(Media Access Control)* Se trata en general de los protocolos de hardware de bajo nivel utilizados para acceder una red en particular. El término dirección MAC se utiliza con frecuencia como sinónimo de dirección física.

MMU (Memory Managament Unit) Componente de un CPU que crea espacios de direccionamiento virtual para cada proceso.

NetBIOS *(Network Basic Input Output System)* NetBIOS es la interfaz estándar para las redes en computadoras personales de IBM y compatibles.

PIC PIC surgió como una abreviatura de *Programmable Integrated Circuit*, pero hoy en día es una marca comercial, y no debe utilizarse como abreviatura. Se divide en familias: PIC12 en la gama baja, PIC16/PIC17 en la gama media y PIC18 en la gama alta.

RS232 Estándar de EIA que especifica las características eléctricas de las interconexiones de baja velocidad entre terminales y computadoras o entre dos computadoras. Aun cuando el estándar más utilizado es RS232C, la mayoría de la gente se refiere a él como RS232

socket Abstracción proporcionada por el sistema operativo UNIX que permite a un programa de aplicación accesar los protocolos TCP/IP

TCP *(*Transmission Control Protocol*)* Protocolo de nivel de transporte TCP/IP estándar que proporciona el servicio de flujo confiable full duplex y del cual dependen muchas aplicaciones.

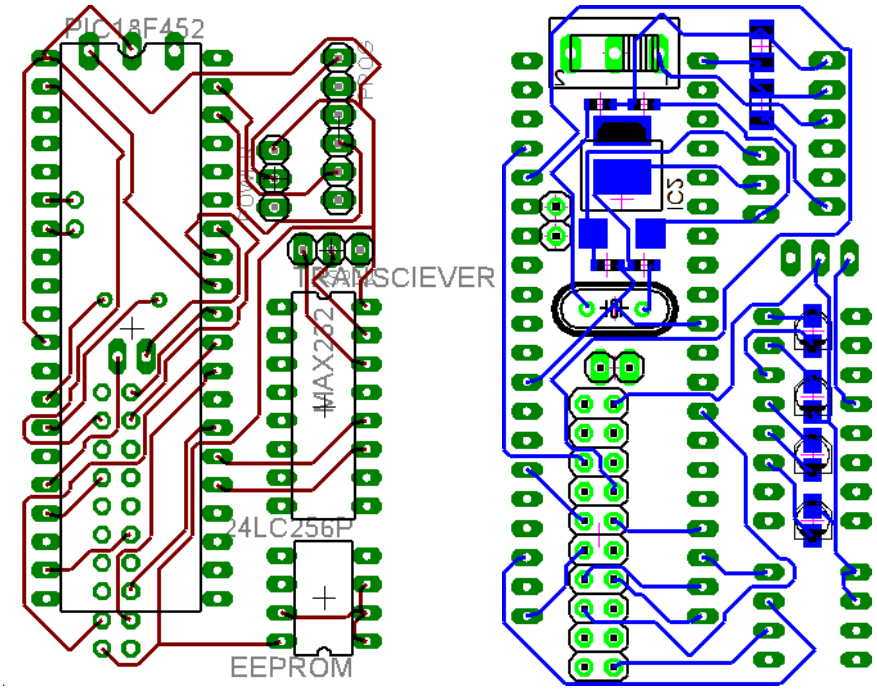
TQFP (Tiny Quad Flat Package) Formato de encapsulado de ciertos circuitos integrados, el cual permite más de 100 terminales en tamaños muy pequeños (menor a $1cm^2$).

UART (Universal Asynchronous Receiver and Transmitter) Dispositivo electrónico que consiste en un solo chip que puede enviar o recibir caracteres en líneas de comunicación serial asíncronas que utilizan RS232.

WAN (Wide Area Network) Cualquier tecnología de red que abarca distancias geográficas extensas.

C. Diseño de PCB

Se utilizó el Software EAGLE, versión 4.09-r2 para diseñar una plaqueta de doble faz. El circuito final fue realizado sobre un sustrato de Pertinax,



(a) Capa superior (b) Capa inferior

Figura C.1.: PCB

D. Esquema electrónico

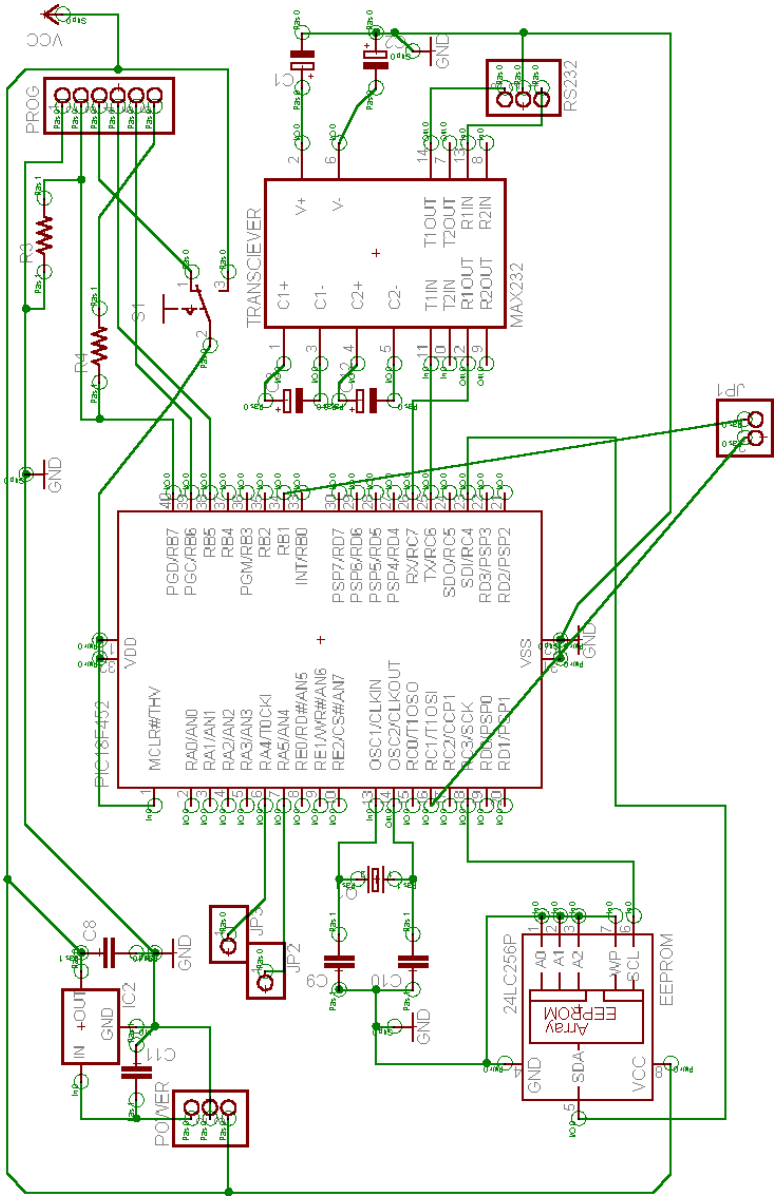


Figura D.1.: Esquema electrónico

E. Imágenes

E.1. Construcción del hardware

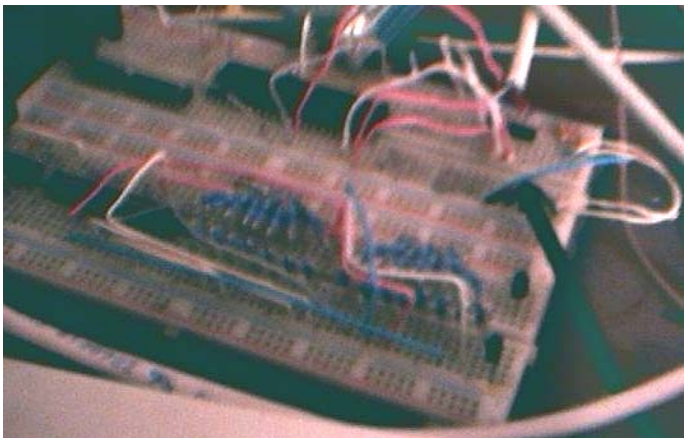


Figura E.1.: Prototipo inicial

Figura E.2.: Confección de la plaqueta

Figura E.3.: Armado inicial

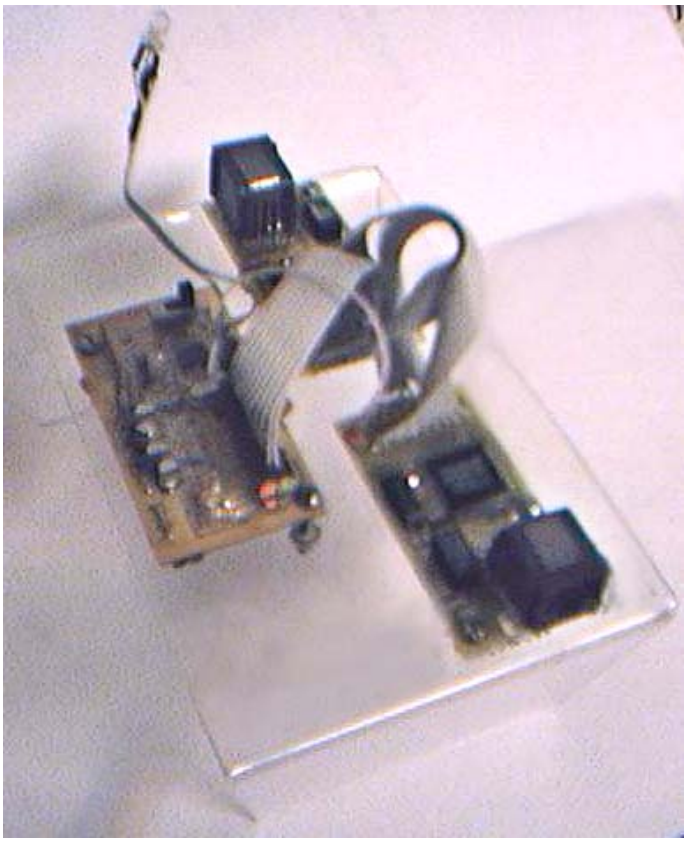


Figura E.4.: Modelo final

E.2. Capturas de pantalla

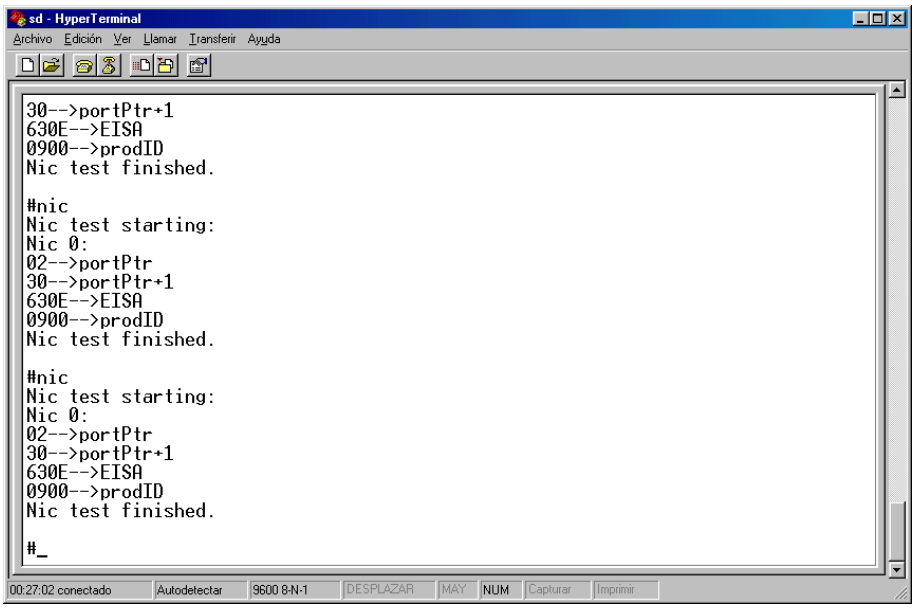


Figura E.5.: Testeos de interfaces en el hyperterminal.

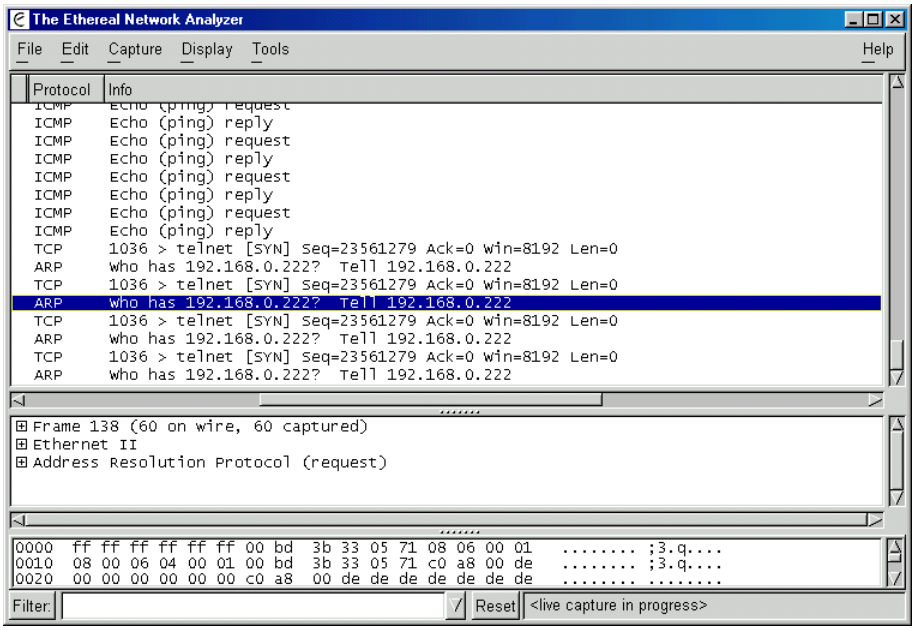


Figura E.6.: Captura sobre Ethereal

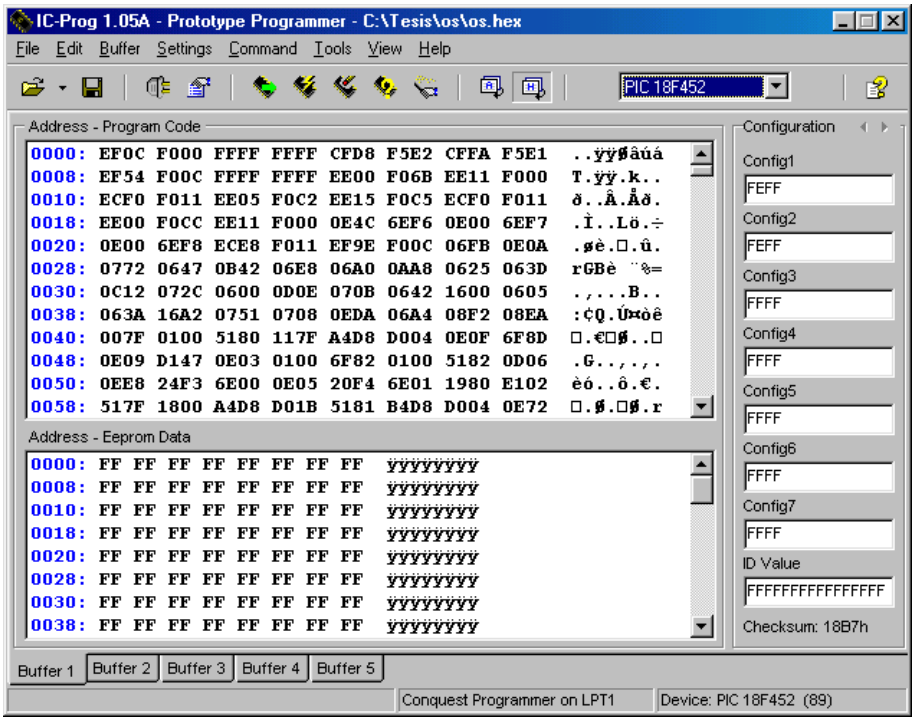


Figura E.7.: Software programador IC-PROG

Índice alfabético