TECNOLÓGICO
DE MONTERREY®

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Ciudad de México

División de Ingeniería y Arquitectura

Departamento de Computación
Departamento de Ingeniería Eléctrica y Electrónica

Ingeniería en Sistemas Electrónicos

flash2web

Diciembre 2004

AUTORES
Mónica Lara
Pablo Enríquez
Ludwig Méndez
Rafael Tello

ASESORES
M. en C. Israel Macías Hidalgo
Dr. Bárbaro Ferro

PROFESOR
Dr. Flavio Lucio Pontecorvo Cassereau

Presentación

flash2web es una interfase de comunicación entre una memoria Flash (como Secure Digital, Memory Stick, Multimedia Card, CompactFlash, o una 'pluma USB') y la red Ethernet vía web, utilizando un dispositivo embebido, y su desarrollo y defensa son el Proyecto de Ingeniería Computacional de Mónica Lara, Pablo Enríquez, Ludwig Méndez y Rafael Tello. Los objetivos de **flash2web** son: Implementar un servidor http y cliente DHCP en un dispositivo embebido; generar el hardware y software necesarios para acceder al sistema de archivos de la memoria Flash; y desarrollar una interfase para comunicar el dispositivo de acceso a Ethernet con el sistema de acceso al medio de almacenamiento. **flash2web** está basado en el Rabbit RCM3200, tarjeta de desarrollo con características avanzadas, gran espacio de memoria y versatilidad en su programación, que se consideraron importantes para su desarrollo.

Tabla de contenidos

1. Introducción

1.1. Situación actual

Hoy en día, se han popularizado enormemente los dispositivos de almacenamiento masivo de estado sólido y la memoria Flash ha ocupado el segmento más importante de este mercado, ya que ofrece ventajas como las siguientes:

- Mayor velocidad que una memoria E2PROM convencional
- Menor tamaño que otros dispositivos de acceso masivo (ZIP drive, discos duros, etc.)
- Al no tener piezas mecánicas, mayor resistencia a golpes o cambios de temperatura
- Costo por megabyte cada vez menor (aún muy alto con respecto a un disco duro).
- Fácilmente adaptable a propósitos con mayor demanda en el mercado (reproductores de MP3, cámaras digitales, Pocket PC's, y memoria de propósito general)

Es fácil darse cuenta de que debido al gran potencial de estas tecnologías es una gran oportunidad poder trabajar con ellas para el desarrollo de nuevos sistemas. Hoy en día la memoria Flash existe en varias presentaciones, y la implementación más común es la de una memoria serial. Existen varios estándares para conectar las memorias Flash seriales con otros dispositivos, entre los cuales destacan el USB (lo que se conocen como las 'plumas USB'), Secure Digital (SD), CompactFlash (CF), Multimedia Card (MMC) y Memory Stick. En todos los casos con excepción de USB, se presentan en forma de pequeñas tarjetas insertables en una ranura especial (dentro de las cámaras digitales, computadoras, reproductores MP3, handhelds, etc). Una de las desventajas de esta tarjeta es el acceso remoto a la información que contiene: es imposible acceder a la tarjeta de memoria desde un punto remoto de la red si la tarjeta no está insertada en un equipo que por un lado tenga la ranura especial, y por el otro tenga capacidades para conectar la tarjeta a la red.

Se ha descuidado un punto relevante de las tarjetas de memoria el sentido de la necesidad de estos tiempos de compartir cierta información que poseen. Estas tarjetas son muy útiles para aplicaciones de un solo usuario, pero si se requiere que la información pueda ser compartida y que pueda utilizarse como multiusuario la tarjeta de memoria no ofrece tantas ventajas. Y falta considerar la compatibilidad, donde muchas de las computadoras no son compatibles con este tipos de memorias y se necesitaría hardware y software extra para lograr compatibilidad. Esta problemática dio pie a imaginar un sistema que brindara un acceso remoto a una tarjeta de memoria sin la necesidad de que esta se encuentre insertada a una computadora por medio de una red Ethernet, y de esta forma también se podrá compartir la información de la tarjeta a todos los usuarios que la requieran sin necesidad de que la computadora del usuario sea compatible con la tarjeta.

1.2. Objetivo general

Desarrollar una interfase entre una memoria Flash serial y el Internet, permitiendo desde un explorador web visualizar y descargar el contenido de la misma, sin emplear las características de lectura de los sistemas operativos, sino a través del acceso directo a la memoria Flash.

1.3. Objetivos específicos

1. Implementar un dispositivo con acceso a Ethernet que sea cliente DHCP y servidor http
2. Generar el hardware y software necesarios para acceder al sistema de archivos de una memoria Flash, desde la interfase física con la tarjeta hasta el soporte para abstraer el sistema de archivos. Se considera para el cumplimiento de este objetivo el uso de memoria Secure Digital, Multimedia Memory Card, CompactFlash, o basada en alguna tecnología USB.
3. Desarrollar una interfase para comunicar el dispositivo de acceso a Ethernet con el sistema de acceso al medio de almacenamiento. Para el cumplimiento de este objetivo se considera válida la implementación de un servidor web de navegación virtual.

1.4. Alcances y limitantes

1. A pesar de que la implementación se llevó a cabo de tal forma que sea utilizable cualquier tipo de partición FAT, flash2web sólo soportará el acceso a FAT16.
2. El acceso al sistema de archivos será de sólo lectura, debido a la complejidad del sistema de archivos FAT16.
3. Únicamente se soportará el formato DOS 8.3, no el Windows Extended Names Format (255.8)
4. Únicamente se soportará el comando GET de http, PUT y POST no serán soportados y su uso podría generar errores desconocidos.
5. La funcionalidad de negociación de IP será únicamente de cliente DHCP o la asignación de una IP estática clase A privada.

2. Marco teórico

2.1. File Allocation Table (FAT)

El sistema de archivos FAT organiza los datos en discos fijos y removibles, accesibles hoy en día en prácticamente cualquier sistema operativo. La principal característica de todas las versiones de FAT (FAT12, FAT16 y FAT32) es su convención de nombrado de archivos, donde se componen de un nombre de un máximo de 8 caracteres y una extensión de 3 caracteres, sin distinción de mayúsculas y minúsculas [6]. En la implementación de FAT de Microsoft, disponible en sus sistemas Windows® y MS-DOS®, la convención de la ruta de acceso para todo archivo se compone de una letra que describa la unidad (o partición) deseada, la ruta de directorios necesaria para llegar al archivo, y finalmente el nombre del archivo, en la siguiente forma:

UNIDAD:\[directorios]\archivo.ext

En toda unidad de almacenamiento para computadora, el espacio disponible de almacenamiento se divide en sectores con un determinado número de bytes. Por su lado, el sistema de archivos FAT16 se basa en dividir la información en clústeres, que es un conjunto de sectores cuyo tamaño está en relación al tamaño mismo del disco o de la partición, como se muestra en la siguiente tabla:

Tamaño de partición	Tamaño de clúster
0 a 32 MB	512 bytes
33 a 64 MB	1 KB
65 a 128 MB	2 KB
129 a 256 MB	4 KB
257 a 511 MB	8 KB
512 a 1023 MB	16 KB
1024 a 2047 MB	32 KB
2048 a 4095 MB	64 KB

Tabla 2.1.1. Tamaños de clúster, tomado de [8].

Es importante destacar que, dado que con 16 bits se pueden direccional 65536 clústeres con un tamaño máximo de 32K (32768 bytes), FAT16 soporta un tamaño máximo de partición de 2GB (2,147,450,880 bytes), aunque por la misma estructura de la misma, el espacio disponible para el usuario puede ser ligeramente menor. De igual forma, un archivo podrá ser de hasta un poco menos de 2 GB (ya que solamente se puede disponer de un máximo de 65,517 clústers para encadenar archivos) [8].

El sistema de archivos realiza un mapeo de cada uno de los clústeres en una tabla de ubicación de archivos (precisamente FAT), donde se proporciona información sobre el clúster, que puede

ser que está disponible para su uso, la dirección del siguiente clúster utilizado para almacenar un archivo o directorio, que está dañado o que representa el final de un archivo.

Un sistema de archivos FAT16 está compuesto usualmente de cinco diferentes secciones:

2.1.1. Master Boot Record (MBR)

El Master Boot Record no forma parte como tal de la partición FAT, sino es una estructura inherente a la arquitectura de las PC's, y es común para la mayoría de los sistemas operativos. En total ocupa 512 bytes, es decir, el clúster 0, sector 1 del disco duro, y contiene el conjunto de instrucciones más básico que ejecuta el procesador de la computadora al pasar el control del BIOS a la unidad de almacenamiento de arranque (hoy prácticamente siempre un disco duro) [9].

Offset	Descripción	Tamaño
000h	Código ejecutable de arranque	446 Bytes
1BEh	Entrada de primera partición (sección 2.1.1.2)	16 Bytes
1CEh	Entrada de segunda partición	16 Bytes
1DEh	Entrada de tercera partición	16 Bytes
1EEh	Entrada de 4ª partición	16 Bytes
1FEh	Marca de ejecutable	2 Bytes

Tabla 2.1.1.1. Master Boot Record, tomado de [6].

2.1.2. Boot Sector

El sector de arranque de cada partición se encuentra siempre en el primer sector asignado a la misma. En el caso de unidades con sólo una partición, este boot sector se encontraría en el sector 2 del disco. Para los fines del presente proyecto, este boot sector es el offset 0 de todos los cálculos realizados, dado que se monta la partición entera, sin incluir el MBR de la unidad.

Parámetro (tamaño)	Dirección
Instrucción de salto (CHAR(3))	00h-02h
Datos del fabricante (CHAR(8))	03h-0Ah
Bytes por sector (WORD)	0Bh-0Ch
Sectores por clúster (BYTE)	0Dh
Sectores reservados (WORD)	0Eh-0Fh
Número de FAT's (BYTE)	10h
Número de entradas raíz (WORD)	11h-12h
Sectores por FAT (WORD)	16h-17h

Tabla 2.1.2.1. Principales datos encontrados en el sector de arranque, tomado de [6]

A partir del boot sector, se calculan las ecuaciones para obtener el offset de cada una de las regiones de la partición de la siguiente forma:

Offset	Dirección absoluta en bytes	Para 128 MB
Inicio	0	0h
FAT1	Inicio + sectores reservados * bytes por sector	1000h
FAT2	FAT1 + sectores por FAT * bytes por sector	20000h
Raiz	FAT2 + sectores por FAT * bytes por sector	3F000h
Datos	Raiz + 32 * Entradas raíz	43000h
Clúster N	Datos + (N-2)*sectores por clúster * bytes por sector	Varía
FAT-Clúster N	FAT1 + (N-2)*2	Varía

Tabla 2.1.2.2. Offsets de las diferentes regiones FAT a partir del boot sector, tomado de [6]

2.1.3. Región de FAT

La región de FAT es precisamente el espacio donde se mapea individualmente cada uno de los clústeres existentes en la partición en una tabla donde se indica el status de cada uno de los clústeres. En el caso de FAT16, esta región es de 2 bytes por cada clúster, y puede contener una de las siguientes palabras:

- **0x0000**: el clúster está libre
- **0x0001 – 0xFFEF**: La dirección del siguiente clúster en la cadena de un archivo
- **0xFFF0 – 0xFFF6:** Clúster reservado
- **0xFFF7:** Clúster dañado
- **0xFFF8 - 0xFFFF**: Fin de cadena de archivo [6].

Debido a que un clúster es de tamaño reducido en un disco, FAT permite el encadenamiento de varios clústeres para almacenar archivos grandes. Este encadenamiento inicia en la entrada de directorio que hace referencia al archivo (ver sección 2.1.4), y en cada entrada FAT de la partición se encuentra la dirección del siguiente clúster o bien el indicador de fin de cadena.

2.1.4. Región del Directorio raíz (Root directory)

La región del directorio raíz comparte la misma estructura que la región de datos; sin embargo se considera una región separada ya que siempre tiene una dirección base fija, y por lo tanto solamente a partir de esta región es posible navegar por todo el sistema de archivos.

En esta región se encuentran las entradas de directorio de todos los subdirectorios y archivos que se encuentran en el directorio raíz. Estas entradas pueden dividirse hoy en día en 3 categorías fundamentales: entradas de subdirectorio, entradas de archivo y entradas LFN. Las entradas LFN son la base de la extensión de Microsoft Windows® que permite utilizar nombres de hasta 255 caracteres de largo y no solamente ASCII, sino cualquier caracter Unicode (chino, árabe, hebreo, etc están incluidos en Unicode) [20]. Sin embargo, el uso de LFN se encuentra

fuera de los alcances del proyecto, por lo que para más información sobre el tema remítase a [20].

Estas entradas tienen un tamaño de 32 bytes, y para FAT16 algunos de los campos con los que cuentan se muestran en la tabla 2.1.4.1. La diferencia fundamental entre una entrada con un directorio y un archivo, es que la del directorio indica un tamaño de 0, y dentro de sus atributos está el ser Directorio. Mientras el número de clúster para una entrada de archivo indica el clúster donde inicia el contenido del archivo, en una entrada del directorio indica el número de clúster donde se encontrarán a su vez las entradas de 32 bytes de ese subdirectorio.

Parámetro	Tamaño
Nombre de archivo	8 bytes
Extensión	3 bytes
Atributos del sistema	1 byte
Fecha y hora de creación	2 bytes
Fecha de actualización	2 bytes
Número de clúster	2 bytes
Tamaño del archivo	4 bytes

Tabla 2.1.4.1 Información encontrada en las entradas de directorio, con información de [20]

2.1.5. Región de Datos

La región de datos contiene todos los archivos y/o directorios referenciados en el directorio raíz, y es propiamente dicho el espacio para usuarios. Una consideración importante en FAT16 y FAT32 (que es su mayor limitante), es que la unidad mínima de asignación de FAT es el clúster, por lo que archivos menores al número de bytes que tiene el clúster ocuparán el clúster entero [6]. Esto tiene como consecuencia que la capacidad del disco quede desperdiciada: en FAT16 basta tener 65517 archivos de 1 byte para llenar el disco duro, aunque este sea de 2GB. Esta consideración es importante para propósitos de diseño en el sentido de que para archivos menores al tamaño del clúster hay que llevar un conteo contra el tamaño del archivo.

2.2. Serial Peripherial Interface (SPI)

Es un estándar de bus serial que fue establecido por Motorola. Las interfaces SPI están disponibles para una gran cantidad de procesadores de comunicación como son algunos microcontrolodadores. SPI maneja un enlace de datos síncrono que opera en full-duplex, o sea que las señales de datos entre los dos dispositivos a comunicarse van en ambas direcciones simultáneamente [14].

Dichos dispositivos guardan una relación maestro-esclavo, donde el maestro inicia el proceso de comunicación. Cuando el maestro genera una señal de reloj y selecciona a un dispositivo esclavo, los datos pueden ser transferidos en uno o ambas direcciones al mismo tiempo.

SPI utiliza cuatro señales de control para lograr la comunicación entre el maestro y el esclavo:
señal de reloj (SCLK).
salida de datos del maestro, entrada de datos del esclavo(MOSI, master data output, slave data input).
entrada de datos del maestro, salida de datos del esclavo(MISO, master data input, slave data output).
señal de selección del dispositivo esclavo.

En la Figura 1, se muestra una configuración de un solo esclavo. Se puede apreciar como la señal de reloj es generada por el maestro y enviada a todos los esclavos. MOSI es una señal que lleva el dato desde el maestro hasta el esclavo, y MISO lleva el dato del esclavo al maestro. El dispositivo esclavo es seleccionado por el maestro cuando la señal de CSS se habilita.

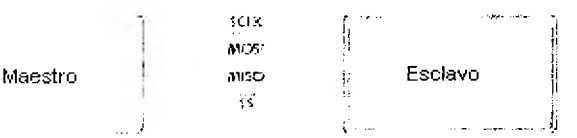


Figura 4.2.1: Diagrama Maestro Esclavo

Si es una configuración de múltiples esclavos entonces el maestro generará una señal de selección para cada esclavo, como se muestra en la figura 2.

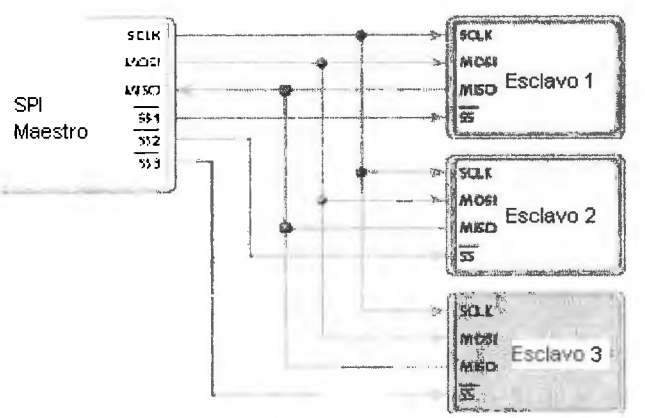


Figura 4.2.2: Multiples Esclavos en modo SPI

Existen dos parámetros llamados clock polarity (CPOL) and clock phase (CPHA), polaridad del reloj y fase del reloj. Estos determinan el flanco de la señal de reloj en que los datos son llevados. Cada uno de estos parámetros tiene dos posibles estados, que permiten 4 posibles combinaciones, cada una de estas son incompatibles entre sí [15].

Por lo tanto, un maestro y esclavo deben usar el mismo parámetro para lograr una comunicación exitosa. Si fuera el caso de una configuración de múltiples esclavos que operan en parámetros diferentes entonces el maestro deberá reconfigurar su parámetro para ajustarse con el parámetro del esclavo con que establecerá el enlace.

SPI no tiene un mecanismo de reconocimiento para confirma la recepción del dato. De hecho, sin un protocolo de comunicación el maestro del SPI no tendría conocimiento de la existencia del esclavo. SPI tampoco tiene control de flujo de datos.

2.3. Secure Digital Memory Card

2.3.1. General

Las memorias Flash poseen gran capacidad de almacenamiento y acceso rápido, combinado con un diseño compacto, lo que hace a este tipo de memoria una opción muy atractiva para ser utilizada en una amplia gama de productos y aplicaciones.[10]

Dichas memorias son memorias no volátiles que tienen las características de memorias RAM, que pueden reescribir datos eléctricamente; y características de memorias ROM, que pueden retener datos aún cuando se les retire la energía.
Las memorias flash tienen tiempos de lectura de alta velocidad y tienen una mayor densidad de memoria debido a su estructura de celdas sencilla.

La memoria Flash es un tipo de memoria EEPROM (Electronic Erasable Programable Read Only Memory), que permite que múltiples localidades sean borradas o escrita en una operación. Una EEPROM normal sólo permite que una localidad de memoria sea borrada o escrita a la vez, así que la memoria flash es más rápida cuando es usada para leer o escribir diferentes localidades al mismo tiempo.

Es un dispositivo de almacenamiento que integra una alta seguridad en manejo de datos y en el almacenamiento de los mismos. Como una memoria Flash, puede almacenar datos aún no tenga suplemento de poder y se pueden realizar actividades de lectura y escritura a una gran velocidad. La figura siguiente muestra la distribución física de los pines así como la asignación de cada señal en su modo estándar.

La tarjeta SD posee un mecanismo de protección para la seguridad de acceso siempre que sea requerido, que funciona de la siguiente manera [11]:

El acceso a la tarjeta debe ser habilitado por medio de autenticación entre los dispositivos.

Un número aleatorio es generado cada vez que hay un proceso de autenticación y se realiza un intercambio de información de seguridad.

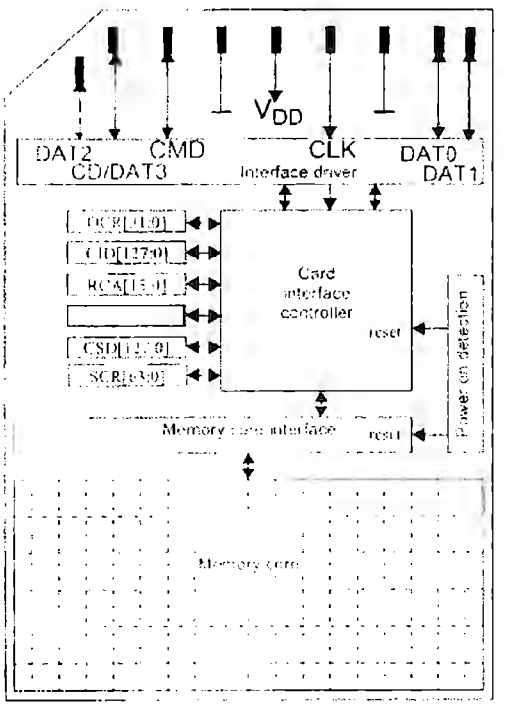


Figura 4.4.1.1: Distribución física de pines usando protocolo SD [5]

2.3.2. Características Principales del Sistema [5]

2.3.2.1. Rangos de voltajes:

- Acceso a Memoria y otros comandos: 2.7-3.6 volts
- SDLV Memory Card (low voltage): Rango de voltaje de operación: 1.6 volts-3.6 volts.

2.3.2.2. Operación:

- Clock rate de 0-25 MHz
- Lectura/Escritura de hasta 10MBytes por segundo al usar 4 líneas en paralelo
- Corrección de errores de campos de memoria

- El remover la tarjeta durante la operación nunca dañará el contenido de la tarjeta
- Protección con el uso de password(opcional)
- Detección de la tarjeta al ser insertada o removida
- Comandos específicos de aplicación

2.3.2.3. Atributos del protocolo del canal de comunicación:

- Canal de comunicación formado por seis pines: clock,command y 4 líneas de datos.
- Transferencia de datos con protección de errores
- Transferencia de datos orientada a un solo bloque o a múltiples bloques.

2.3.2.4. Comunicación externa

La comunicación en la SD Memory Card está basada en una interfase de de 9 pines, diseñada para operar en una frecuencia máxima de 25MHz. Los 9 pines son: 1 pin de reloj (CLK), 1 pin de comando (Command), 4 pines de Datos, y 3 pines de Líneas de Poder.

La SD Memory Card soporta dos tipos de protocolos de comunicación, el SD y el SPI.
La aplicación es capaz de elegir el modo de comunicación deseado. El modo de operación es detectado automáticamente mediante el comando de RESET para que la comunicación pueda ser realizada en este modo [5].

2.3.3. Comunicación usando el bus SPI

El protocolo de comunicación SPI que tiene esta tarjeta es compatible con el modo SPI encontrado en la mayoría de los microcontroladores. El estándar SPI define el enlace físico de la tarjeta como un sistema esclavo con un microcontrolador como sistema maestro. Mayor información sobre SPI en general, puede ser encontrado en la sección precedente.

Como cualquier otro dispositivo SPI, el canal SPI de la SD consiste de las siguientes señales:

- CS: Señal de Chip Select (Host-Tarjeta)
- CLK: Señal de reloj (Host-Tarjeta)
- Data In: Señal de datos (Host-Tarjeta)
- Data Out: Señal de datos (Tarjeta-Host)

Debido a la especificación que establece la norma de comunicación la SD en su modalidad de SPI se crea la siguiente configuración eléctrica cumpliendo con los requisitos de alimentación.

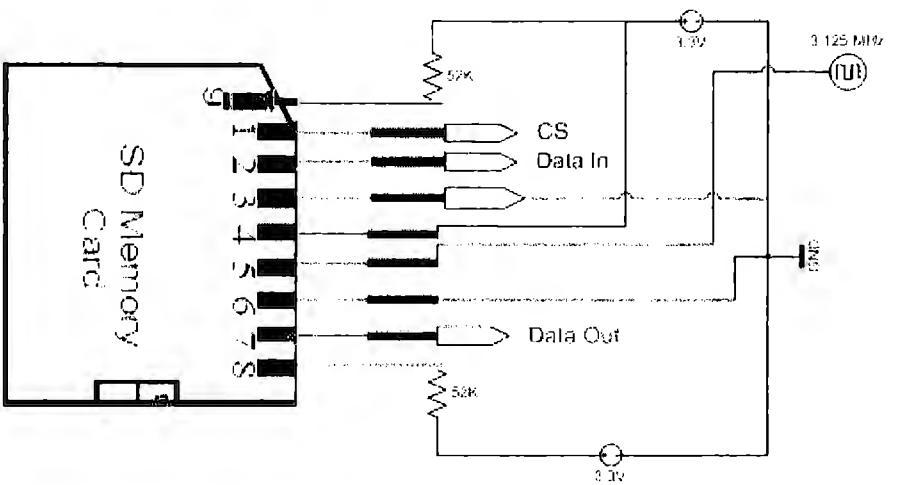


Figura 2.3.3.1: Descripción eléctrica para SD

Para poder lograr una comunicación con la tarjeta se debe tener en cuenta que existen diversos comandos y respuestas para lograr la comunicación deseada, también se debe de tomar en cuenta que para realizar una operación de inicialización, lectura o escritura se tiene definido una secuencia de cómo recibir y enviar datos. En el caso de este diseño se utilizaron los siguientes comandos y secuencias de operaciones:

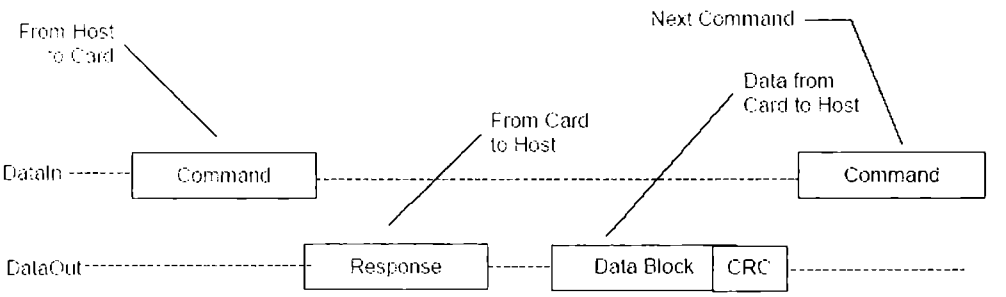


Figura 2.3.3.2: Operación de Lectura [5]

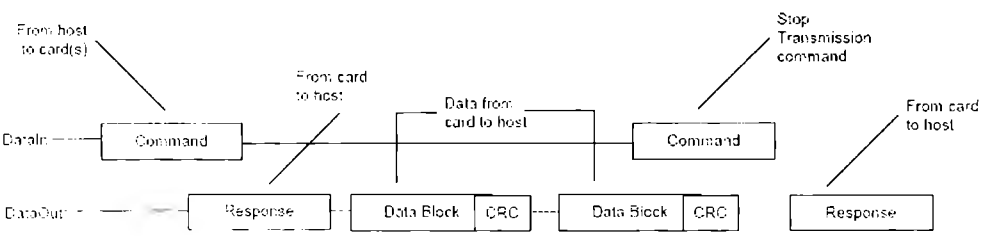


Figura 5.2.2.2: Lectura de Múltiples Bloques

Comando	Atributos	Respuesta	Acción
0	Ninguno	R1	Resetea la tarjeta

16	31:0	R1	Define el valor del bloque con que se realizaran operaciones (lectura y escritura)
17	31:0	R1	Lee la el contenido de la dirección especificada en el atributo
18	31:0	R1	Lee continuamente datos a partir de la dirección especificada hasta que reciba una señal de paro
12	Ninguno	R1b	Detiene la transmisión continua

Tabla 2.3.3.1. Comandos de la tarjeta SD [5]

La comunicación a la SD usando SPI requiere de ciertas consideraciones, ya que por defecto la SD se encuentra en modo de comunicación Standard (Protocolo SD), para ello se debe de tener una secuencia de inicialización que la pondrá en modo SPI.

- Modo de inicialización para protocolo SPI
- Poner la señal CS en nivel bajo para la SD.
- Enviar el comando 0 a la tarjeta
- Esperar respuesta R1 de la tarjeta.

Una vez inicializada en modo SPI la tarjeta seguirá de esta forma hasta que deje de tener alimentación en sus terminales.

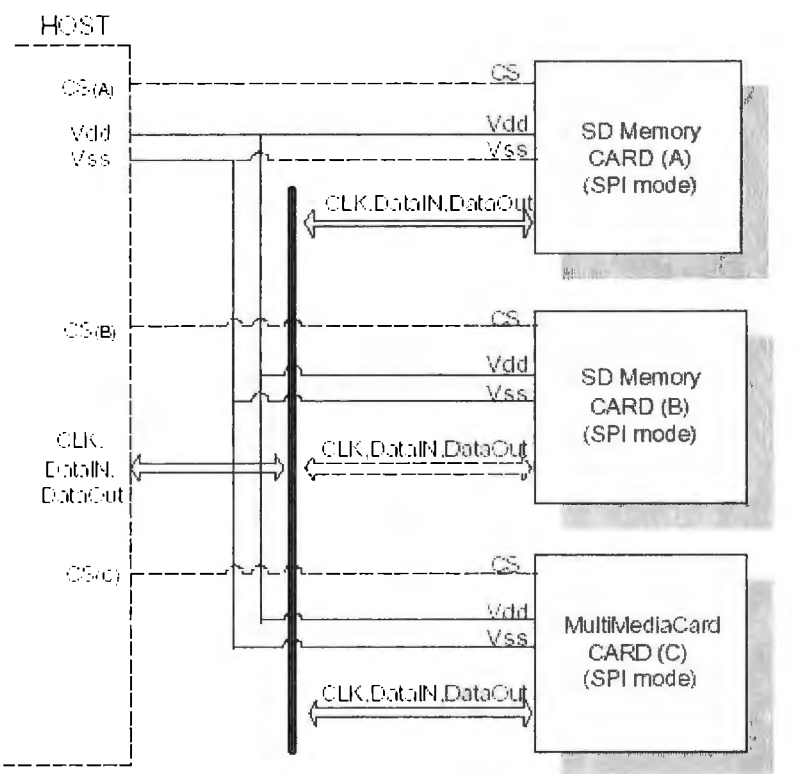


Figura 2.3.3.3: Bus para conexión de múltiples SD usando SPI [5]

Otra característica común de la interfase SPI de la SD es la transferencia se realizara en bloques desde 1 hasta 4 bytes según las palabras de control que se le envíen a la tarjeta.

Los métodos de identificación y de direccionamiento son reemplazados por una señal de hardware de CS (chip select). No hay comandos de broadcast. Para cada comando la tarjeta es seleccionada o habilitada (línea en bajo) utilizando la señal de CS. Esta señal debe ser continuamente activada durante todo el tiempo de la transacción. La transacción consta del comando, la respuesta y el envío del dato. La excepción para esta regla del CS (habilitado) es durante la el proceso de programación [5].

La interfase SPI tiene una distribución diferente de pines a la del estándar SD, ya que SPI utiliza un número menor de pines y la transmisión no es paralelo. Por lo anterior se define la siguiente estructura:

Pin	*Nombre*	*Tipo de PIN*	*Descripción*
1	CS	Entrada	Selector de Chip (Activo en bajo)
2	Data In	Entrada	Receptor de datos y comandos provenientes del master
3	VSS1	Alimentación	Tierra.
4	VDD	Alimentación	Voltaje de alimentación (2.7 -3.6V)
5	CLK	Entrada	Entrada de reloj por parte del master
6	VSS2	Alimentación	Tierra
7	DATA OUT	Salida	Salida de datos hacia el master
8	-	-	No se usa, sin embargo se debe conectar con una resistencia de Pull-up.
9	-	-	No se usa, sin embargo se debe conectar con una resistencia de Pull-up.

Tabla 2.3.3.2. Mapa de pines de SPI en la Secure Digital [5]

El canal SPI está orientado a bytes. Cada comando o bloque de datos está construido por 8 bits (1 byte) y está alineado por byte con la señal CS. Los mensajes SPI consisten de un comando, la respuesta y un bloque de datos. Toda las comunicaciones entre la tarjeta y el host serán controlados por el host, por lo que se utilizará un tipo de arquitectura maestro-esclavo. Para iniciar la transacción por el bus de comunicación, el host debe habilita la señal de CS, recordemos que este pin se habilita en bajo.

Existen ciertas características en el comportamiento de la respuesta de la tarjeta:

- La tarjeta siempre responderá al comando.
- Dos estructuras de respuesta pueden ser usadas (8 bits y 16 bits)
- Cuando la tarjeta encuentra un problema de recuperación de datos, la tarjeta reemplazará el bloque de datos esperado con una respuesta de error.

Los diferentes comandos que acepta la SD son los siguientes:

Comando	*Parámetros*	*Respuesta*	*Descripción*
0	Ninguno	R1	Activa un reset por software a la tarjeta.
1	Ninguno	R1	Activa la inicialización de la tarjeta.
9	Ninguno	R1	Pide la información descriptiva de la tarjeta.
10	Ninguno	R1	Pide la identificación de la tarjeta
12	Ninguno	R1b	Forza a detener la trasmisión de múltiples bloques
13	Ninguno	R2	Pide el estatus de la tarjeta.
16	0:31	R1	Selecciona el tamaño de bloque para operaciones de lectura y escritura
17	0:31	R1	Lee un bloque de la dirección que recibe como parámetro.
18	0:31	R1	Lee continuamente bloques de la dirección que recibe como parámetro hasta recibir un stop
24	0:31	R1-3	Escribe un bloque en la dirección que recibe como parámetro.
25	0:31	R1	Continuamente escribe bloques desde la dirección que recibe como parámetro hasta que recibe una señal de stop.
27	Ninguno	R1	Programa bits del registro CSD
28	0:31	R1b	Activa el bit de protección par el bloque definido en la dirección que se pasa como parámetro.
29	0:31	R1b	Desactiva el bit de protección par el bloque definido en la dirección que se pasa como parámetro.
30	0:31	R1	Le pregunta el status de los bloques que están protegidos definidos por la dirección que recibe como parámetro,
32	0:31	R1	Selecciona la primera dirección del bloque se borrara
33	0:31	R1	Selecciona la ultima dirección del bloque se borrara
38	0:31	R1b	Borra los bloques anteriormente seleccionados

55	0:31	R1	Especifica que el siguiente comando será aplicación
56	0:31	R1	Transfiere o envía datos
58	Ninguno	R3	Lee el registro OCR
59	0:31 *	R1	Apaga modo CRC

Tabla 2.3.3.3. Comandos SD [5] y [11]

Las respuestas posibles que se pueden obtener debido a los comandos antes mencionados usando SPI son las siguientes:

2.3.3.1. Formato R1

La respuesta que recibe es de longitud de un byte y el bit más significativo estará puesto en cero mientras que los siguientes bits representan posibles errores cuando su estado es en 1 lógico.

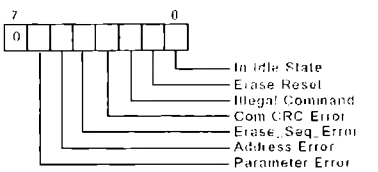
Estado	*Diagrama*
Idle State: la tarjeta esta en modo de espera y esta en proceso de inicialización	
Erase reset: una secuencia de borrado fue detenida antes de que se pudiera ejecutar debido a otro comando de borrado.	
Illegal Comand: Se recibió un comando no valido.	
Communication error: Ocurrió un error en la secuencia de comandos	
Address Error: No existe la dirección que se envio.	
Parameter Error: El argumento esta fuera de rango para esta tarjeta	

Tabla 2.3.3.1.1. Respuesta R1 [5]

2.3.3.2. Formato R1b

Este formato de respuesta es igual al anterior agregando que el bit mas significativo puede tomar dos valores

- 1 - la tarjeta esta lista para el siguiente comando.
- 0- La tarjeta esta ocupada.

2.3.3.3. Formato R2

El formato R2 es de dos bytes, el primer byte representa exactamente la respuesta R1 y el segundo byte se define de la siguiente forma.

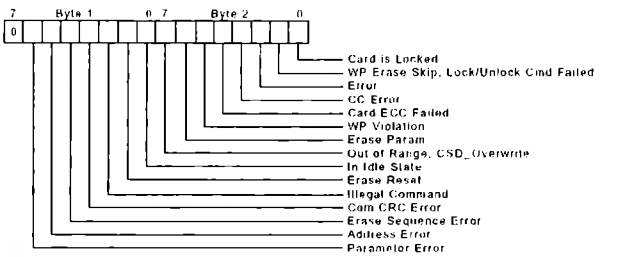
Estado	Diagrama
Erase parameter: selección invalida para sectores a borrar	
Write protec violation: El comando trato de escribir en un bloque protegido	
Card ECC Failed: ECC se aplico sin embargo fallo la correccion del dato	
CC error: Error interno del controlador	
Error: Ocurrio un error	
Write Project erase skip: solo se pudo borrar parcialmente el espacio seleccionado.	
Card is locked: la tarjeta tiene el seguro	

Tabla 2.3.3.3.1. Respuesta R1 [5]

2.3.3.4. Formato R3

Este formato de respuesta esta compuesto de 5 bytes, la estructura del bit mas significativo es idéntico a la respuesta R1, los otros cuatro bytes contienen el registro OCR

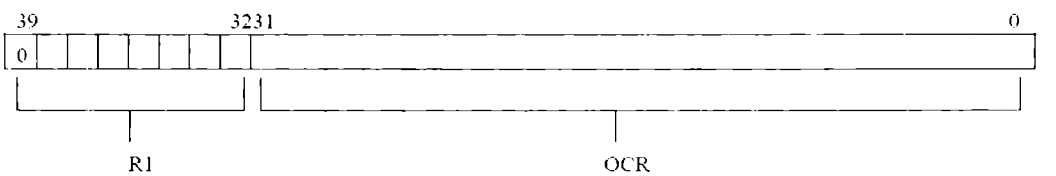


Figura 2.3.3.4.1: Respuesta R3 [5]

2.4. Tecnologías web

2.4.1. Ethernet

Es la tecnología de LAN (Local Area Network) más utilizada en todo el mundo. Está descrita en el estándar 802.3. Ethernet utiliza el método CSMA/CD(Carrier sense multiple access with collision detector) para compartir la red.

El estándar 802.3 permite cuatro velocidades en la red. El estándar original soportaba solamente 10Mb/s. Ahora existen Fast Ethernet, de 100Mb/s; Gigabit Ethernet de 1Gb/s, y 10Gigabit Ethernet, de 10 Gb/s.

Una comunicación por Ethernet puede tomar las ventajas de protocolos de alto nivel como son TCP e IP, pero también puede usar algún protocolo de aplicación específica. Para que exista una comunicación vía Ethernet simplemente se necesita que los dispositivos, como microcontroladores por ejemplo, tengan un chip controlador de Ethernet.

2.4.1.1. Control de acceso al medio

En redes Ethernet que usan interfaces half-duplex, sólo una interfase puede transmitir al mismo tiempo, así que se debe realizar un proceso para tomar la decisión de qué interfase tendrá acceso al medio y en qué momento, evitando colisiones y fallas en la comunicación. Como ya se había mencionado Ethernet utiliza como control de acceso al medio el llamado método CSMA/CD (carrier sense multiple accesss with collision detection).

Existen interfaces ful-duplex en las que el dispositivo puede tanto enviar como recibir datos. Esta es una gran ventaja porque en estos casos un segmento de red está libre de colisiones.

2.4.1.2. Frames

En Ethernet los datos viajan a través de estructuras llamadas "frames". Un frame tiene tanto campos de datos como campos para guardar la información necesaria para que el paquete llegue a su destino, así como para que el receptor del paquete defina si este llegó íntegro.

Por esto los controladores de Ethernet arman la información en frames para que estos sean transmitidos, y también extraen y almacenan la información recibida en los frames.

La información que tienen los frames para que el dato pueda ser transmitido o recibido son los siguientes:

- Bits de sincronización
- Información de las direcciones de fuente y destino
- Secuencia de chequeo de errores
- Información reidentificación del dato que se envió.

2.4.1.3. El protocolo de Internet

En las redes Ethernet es común buscar una comunicación local y también una comunicación con Internet. Se utiliza protocolos de Internet como son TCP, UDP e IP. Estos protocolos se encargan de regular el flujo de datos, de direccionamiento, y del ruteo de la información o mensajes a ser transmitidos. Los mensajes que viajan a través de Internet deben usar el protocolo IP.

Para lograr una comunicación en Internet el dispositivo a conectarse debe de tener tres elementos fundamentales: dirección IP, con propósitos de identificación del dispositivo en Internet; un router que tenga acceso al Internet; y la habilidad de poder trabajar datagramas de IP.

Debido a que un dispositivo que quiere conectarse a la red necesita una dirección IP, esta dirección es proporcionada por el ISP (Proveedor de Servicios de Internet). Hay dos tipos de direcciones IP: estáticas y dinámicas.

Una dirección IP estática permanece siempre hasta que el administrador del dispositivo decide cambiarla. Por otro lado, una dirección IP dinámica puede cambiar cada que el dispositivo es reiniciado o conectado a la red.

Un sistema embebido podría almacenar su dirección IP estática en su memoria no volátil, o por medio de una aplicación, etc. Pero también la dirección IP ya sea dinámica o estática puede ser obtenida del DHCP (Dynamic Host Configuration Protocol) Server.

Un paquete viaja por la red a través de routers que examinan su dirección IP y direccional el paquete hacia su destino. Ya que la maquina ha recibido el paquete, este se descomponen en sus componentes principales y se analiza, el paquete contiene una parte llamada encabezado IP, que dice en qué protocolo, ya sea UDP o TCP, debe recibirse el paquete.

El protocolo de Internet tiene entonces la tarea de definir las direcciones de fuente y destino de un paquete, a través de cualquier interfase de red así como a través de redes que utilizan diferentes interfaces. Por otro lado, se define un protocolo que permite el envío de datagramas que pueden ser fragmentados y viajar por la red, pero luego reensamblados al llegar a su destino.

2.4.1.4. Direcciones IP

Existen varias formas para que un dispositivo obtenga su dirección IP, una de ellas es por medio de DHCP, el cuál define tres formas de lograrlo: obtención manual, automática y dinámica.

Para esto ser requiere de que un dispositivo funcione como servidor de DHCP. Las máquinas que están en la red serán los clientes de DHCP y pedirán su dirección IP al servidor. Este utilizará uno de los tres métodos para la obtención de la dirección IP.

Cuando un cliente se conecta a la red, el cliente de DHCP utiliza UDP para enviar a todas las máquinas de la red un mensaje de DHCPDISCOVER para pedir una dirección IP. Debido a que el dispositivo todavía no tiene dirección IP entonces usa la dirección 0.0.0.0. Y el servidor DHCP le envía la dirección IP correspondiente orientándose con la dirección de hardware.

2.4.1.5. Obtención manual de la dirección IP

Mediante este método el administrador de la red determina la dirección IP de cada host de la red, pero en lugar de configurar dicha dirección directamente en cada host, el administrador configura dicha dirección en el servidor DHCP. Este envía la dirección al host cuando la pide por medio de un mensaje DHCPDISCOVER.

2.4.1.6. Obtención automática de la dirección IP

Mediante este método el servidor de DHCP se encarga de elegir entre una lista de direcciones IP disponibles para ser enviada a host que pide la dirección IP.

2.4.1.7. Obtención dinámica de dirección IP

En este método también se tiene una lista de direcciones disponibles pero se permite que una dirección IP no sea asignada de manera permanente, si no que se asigna dicha dirección al host por un tiempo determinado. Si después de este tiempo el se quiere conservar la dirección entonces el host debe enviar mensajes periódicamente para renovar la petición. Si el host no envía dichos mensajes entonces perderá la dirección IP.

2.4.1.8. Servidores de Páginas Web

Un browser es una aplicación cliente que utiliza HTTP para pedir una página web de un servidor de Internet o de una red local. HTTP(Hyper Text Transfer Protocol) es un protocolo para la transacción de archivos hipertexto en Internet. Define como los mensajes son formateados y transmitidos, y las acciones correspondientes de los servidores de web y browsers en respuesta a ciertos comandos [12].

Un browser provee una interfase al usuario para la petición de páginas web, pero para ciertas aplicaciones de un sistema embebido, este deberá comportarse como cliente HTTP. Un sistema embebido puede funcionar como servidor de una página conteniendo texto e imágenes simples, puede desplegar datos en tiempo real.

Un sistema embebido que funciona como un servidor web tiene generalmente las siguientes características:

- Memoria no volátil, para almacenar las páginas que estarán en el servidor.
- Soporte para TCP e IP. Peticiones para páginas Web y la respuesta de éstas viajará en la porción de datos de los segmentos TCP.
- Soporte para HTTP. El servidor debe ser capaz de entender y responder a las peticiones recibidas por las páginas Web.
- Debe tener conexión a Internet o a una red local.
- Páginas en el servidor. Las páginas Web son archivos o bloques de texto que usan una forma de decodificación llamada HTML, el cual puede incluir ligas e imágenes y links a otras páginas u otros recursos.

2.4.1.9. Servidores de páginas con datos dinámicos.

Muchas páginas Web son estáticas, esto es, que la información de la página no cambia a menos que alguien edite la página Web. Los sistemas embebidos que funcionan como servidores Web generalmente tendrán la función de desplegar información en tiempo real (datos dinámicos), esto es que la información contenida en la página puede estar cambiando. Un ejemplo de esto es la lectura de un sensor.

Cuando se trabaja con datos dinámicos y la página hace una petición al servidor, la aplicación del servidor debe insertar los valores actuales o corrientes de las variables en los lugares correspondientes de la página Web.

2.4.2. Tráfico de información en http

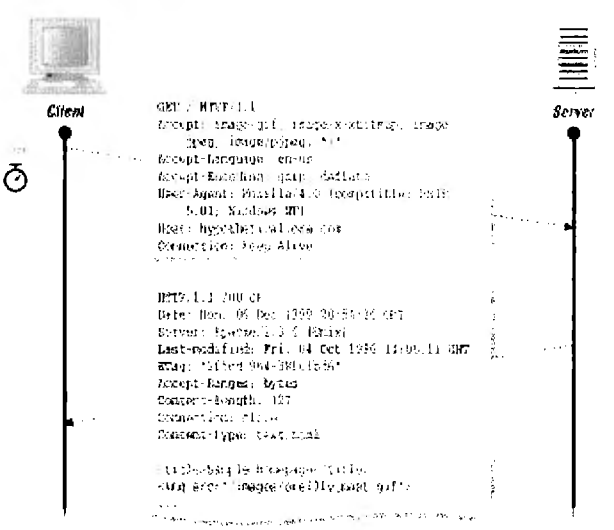


Figura 4.4.5.1. Resumen del tráfico de información

El tráfico de información en HTTP consiste en peticiones y respuestas; es un protocolo sin estados, esto quiere decir, que ni el navegador ni el servidor pueden mantener cuenta de las peticiones previas, para cada petición hay una respuesta y nada más [13].

Los formatos de las peticiones y de las respuestas están estandarizados según la norma RFC 288 de la IETF, que es en realidad una especificación del estándar RFC 2068 donde se definen las características del protocolo HTTP versión 1.1 [13].

Según estos estándares, la forma principal tanto de las peticiones como de las respuestas es:

- Línea Inicial (Diferente entre Petición y Respuesta)
- Cero o más líneas de encabezado
- Una línea en blanco como separador
- El cuerpo del mensaje (Archivo html, datos o respuesta a una forma, etc.)

Puesto de otra forma, el formato de un mensaje HTTP es:

<Línea inicial que difiere entre petición y respuesta><CRLF>
Encabezado 1: Valor
Encabezado 1: Valor
Encabezado 1: Valor

<Cuerpo del mensaje; puede ser un archivo html o datos enviados por el navegador, de muchas líneas de código o simplemente información binaria de un archivo.>

Cada línea debe de terminar con el ASCII 10 ó 13, CRLF ó LF respectivamente. Para explicar como nuestro programa interpretara las peticiones del navegador de Internet y contestara con sus propios mensajes, analizaremos las peticiones y las respuestas por separado.

2.4.2.1. Petición de HTTP

Un ejemplo real de petición HTTP sería por ejemplo:

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE
    5.01; Windows NT)
Host: cursos.itesm.mx
Connection: Keep-Alive
```

- La primera línea indica el recurso que se solicita al servidor, proseguido de la versión HTTP más alta que el cliente soporta.
- La segunda línea comienza con los encabezados, en ella el navegador informa al servidor el tipo de archivos que preferiría recibir. Esta línea será ignorada para fines de nuestro dispositivo.
- La tercera línea define el idioma que maneja el navegador. Esta línea será ignorada para fines de nuestro dispositivo.
- La cuarta línea dice que el navegador puede recibir la respuesta encriptada en gzip. Esta línea será ignorada para fines de nuestro dispositivo.
- La quinta línea identifica al navegador, entre paréntesis dice que el navegador es en realidad Internet Explorer. Esta línea será ignorada para fines de nuestro dispositivo.
- La sexta línea contiene el nombre del host, útil para permitir atender peticiones de diferentes servidores en una misma IP.
- La séptima y última línea de encabezados dice al servidor que debe mantener abierta la conexión hasta que el cliente le pida cerrar. Nuestro dispositivo tentativamente cerrara la conexión de cualquier manera pues esto no afecta en realidad el desempeño cuando la transmisión es solo de un archivo.

La siguiente línea debe de ser en blanco para distinguir de cualquier documento que se esté agregando a la petición, recordando que las peticiones pueden ser para subir archivos y no solo para bajar páginas web y otros documentos varios. El dispositivo ignorará la mayoría de los comandos del encabezado, y sólo deberá confirmar que el comando en cuestión es un GET y, muy importante, qué es lo que el navegador le está pidiendo de la tarjeta SD.

2.4.2.2. Respuesta en HTTP

Una respuesta imaginaria a la petición usada en el ejemplo anterior sería:

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 2004 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>¡Feliz año nuevo!</h1>
(todo el código necesario)


.
</body>
</html>
```

La primera línea incluye el protocolo y versión utilizados en el mismo formato que la petición, junto con un código y una descripción del mismo. Hay 5 tipos de códigos:

1xx	Mensaje informativo
2xx	Éxito en la petición
3xx	Redirección de URL
4xx	Error del cliente
5xx	Error del servidor

Tabla 4.4.5.2.1. Resumen de respuestas http [12]

Los códigos más comunes son:

200 OK

La petición fue exitosa y el recurso solicitado se incluye en el cuerpo de la respuesta.

301 Moved Permanently

302 Moved Temporary

303 See other

El recurso se ha movido a otra locación y el navegador debe de pedir el archivo marcado en el dato "Location: www.XYZ.com" en los encabezados. Se usa mucho para no recurrir a la pantalla de archivo no encontrado del sistema.

404 Not Found

El recurso solicitado no existe

500 Server Error

Hubo un error en el servidor, a menudo es un script con sintaxis incorrecta, y por lo tanto no se puede recuperar el archivo solicitado. [12]

La segunda línea del encabezado da la fecha y hora de la respuesta del servidor.
La tercera línea es la más importante, porque le describe al cliente el tipo de archivo que recibirá, de esta manera el cliente podrá prepararse para manejar el archivo con el programa pertinente y no solo mostrar el código binario en la ventana del navegador. El dispositivo identifica el tipo de contenido de cada uno de los archivos y lo envía junto con el archivo solicitado.
La cuarta línea marca el tamaño del archivo que se adjunta y, aunque no es necesaria, es en general recomendada. En la primera etapa del proyecto no está planeado incluir esta línea, ya que el contenido de la misma es dinámicamente generado.
Después de una línea en blanco viene el archivo adjunto, este puede ser html, texto simple o cualquier tipo ya especificado en el encabezado "Content-Type"; en el caso de archivos binarios su código es agregado y puede incluso verse como caracteres de texto si el navegador no llegara a recibir la especificación de tipo.

2.5. Rabbit RCM3200

El Rabbit RCM3200 es un dispositivo construido a partir del microprocesador Rabbit 3000® , basado en el chip Z180, altamente popular durante la década de los 80's y 90's, en un principio como una seria competencia para Intel. El Rabbit 3000 es un microprocesador que ha sido embebido junto con 512K de memoria RAM, 512K de memoria Flash, un puerto de Ethernet, puertos infrarrojos, RS232, batería de respaldo para la memoria CMOS, para formar un sistema de programación para control basado en Ethernet. El Rabbit puede ser programado directamente en su ensamblador, o bien utilizando compiladores como el Dynamic C, el cual contiene la mayoría de las características del ANSI C. El Dynamic C contiene a su vez un set completo de bibliotecas que permiten el uso eficiente de los dispositivos embebidos en forma muy transparente para el usuario [4].

Este módulo opera a 44.2MHz, tiene una memoria SRAM de datos y de programa, memoria flash, dos relojes(un reloj oscilador principal y otro que funciona como contador de tiempo), circuitería interna encargada del reset y encargada del control del respaldo de la batería del reloj interno en tiempo real y memoria de datos SRAM del microprocesador. El RCM3200 utiliza 3.3 volts los cuales recibe de la tarjeta madre en la que está montada, también puede conectarse con todo tipo de dispositivos digitales compatibles CMOS, por medio de la tarjeta madre [2].
Algunas de sus características principales son las siguientes [4]

- Microprocesador Rabbit 3000 a 44.2MHz
- 52 líneas paralelas de entrada o salida de 5 V : 44 configurables para entrada o salida, 4 fijas para entradas y 4 fijas para salidas.
- Dos salidas digitales adicionales, dos entradas digitales adicionales.

- Entrada externa de reset.
- Bus alterno , que puede ser configurado para 8 líneas de datos y 6 líneas de dirección(compartido con las líneas paralelas de entrada y salida), líneas de entrada/salida y lectura/escritura.
- Diez temporizadores de 8 bits y un temporizador de 10 bits.
- Reloj de tiempo real
- Supervisor watchdog
- Puerto Ethernet 10/100Base-T RJ-45
- Contador PWM de 10 bits y corrida libre
- Captura de entrada de dos canales que puede ser usada para señales de tiempo de entrada por medio de varios pines de puerto.
- Decoder de Cuadratura de dos canales que puede aceptar entradas de módulos de encoder incremental externo.
- Seis puertos seriales compatibles con CMOS: tiene un baud rate máximo asíncrono de 5.5. Mbps. Cuatro puertos son configurables como puertos SPI, y dos son configurables como puertos seriales SDLC/HDLC.
- Soporta un transmisor-receptor IrDA de 1.15 Mbps

2.5.1. El Rabbit RCM3200 como servidor web

El uso de una página web dinámica en C, en forma tradicional resulta bastante complicada; ya que el control de todos los aspectos propios del entorno web se tiene que hacer en forma totalmente manual. Sin embargo, Dynamic C ofrece 2 bibliotecas, TCP.LIB y HTTP.LIB , que permiten realizar las principales acciones que se requieren en un servidor web, entre las que destacan [2]:

- Asignación de IP en forma estática o dinámica a través de un servidor DHCP
- Manejador predeterminado para diversos tipos de MIME
- Parseo de archivos SHTML en forma nativa, logrando escribir variables de C en HTML utilizando únicamente un programa como Dreamweaver
- Línea de tiempo del programa configurable en función a eventos (una cierta acción ocasiona la llamada a una función en C), como en Visual Basic y C#.
- Posibilidad de incluir archivos anexos (como GIF, JPG, SWF, etc.) directamente desde el tiempo de diseño.

3. Descripción del proyecto

3.1. Diagrama a bloques

3.2. Resumen

Para implementar la lectura de FAT16 en flash2web se desarrolló en primer lugar un algoritmo general de acceso en C basado en estructuras de datos estáticas (debido a las limitaciones del propio Rabbit). Para probarlas se realizó un programa que emula el núcleo (Kernel) de un sistema operativo, brindando acceso a una partición FAT16 virtual contenida en un solo archivo ordinario. Con comandos básicos de UNIX y DOS, es posible realizar operaciones básicas (como copiar al sistema real de archivos, visualizar, cambiar de directorio y mostrar el directorio actual) sobre este sistema virtual.

El acceso a la memoria Flash se desarrolló por varias líneas de investigación. Por un lado, se realizó la conexión de una tarjeta Secure Digital a través de un puerto serie síncrono (SPI) del Rabbit. Sin embargo, debido a que se descubrió que el Rabbit tiene algunos problemas con la implementación de SPI al requerir de pines adicionales para conmutar el tercer estado del pin MISO, se creó un emulador de memoria Flash en C#.NET, que se conecta de la PC a la tarjeta Rabbit a través de un chip USB a puerto serie asíncrono. Este emulador responde a los comandos del Rabbit de la misma forma que una tarjeta Secure Digital, con tiempos de lectura y escritura aproximadamente iguales.

El cliente DHCP fue implementado utilizando las bibliotecas del propio Rabbit. Si después de 60 segundos de solicitar IP no hay respuesta del servidor, se asigna estáticamente una IP. Por su lado, el servidor web tiene 2 funciones principales: mostrar el árbol de directorios a partir de las estructuras generadas por el algoritmo FAT, y permitir la descarga de archivos. La navegación entre directorios se hace mediante paso de parámetros GET hacia el servidor, quien solicita los cambios de directorio al sistema FAT. Cuando se solicita un archivo en específico, flash2web reconoce algunos tipos de archivo de texto, mismos que muestra en pantalla; los demás tipos MIME son extraidos de la partición y copiados a un buffer desde donde el explorador web los puede descargar; con ello, la lógica de qué hacer con el archivo se deja al navegador. Para probar este funcionamiento se desarrolló una versión modificada del primer emulador, que en

lugar de recibir parámetros y devolver resultados utilizando la consola MSDOS, se hace al cliente web utilizando IIS sobre la plataforma Windows XP®. A diferencia del primer emulador, en este no se puede guardar un 'estado de sesión', por lo que se requieren enviar los parámetros suficientes para poder cada vez que se invoca, abrir el archivo binario, realizar todas las operaciones necesarias, generar los resultados y terminar el programa.

Finalmente, se integró toda la funcionalidad probada de los emuladores en un solo programa para ejecutarse en el Rabbit, que tuviera acceso al emulador C#.NET en forma invisible utilizando el puerto serial asíncrono, control sobre el sistema de archivos FAT, servicio de cliente DHCP y servidor web que realice todas las funciones anteriormente mencionadas.

3.3. Software

3.3.1. Funciones y estructuras comunes

Con excepción del emulador de memoria serial (3.3.5), todos los programas utilizan un conjunto de funciones comunes, que definen las lecturas de acceso a la tarjeta Flash. Si bien algunas mejoras fueron hechas de último momento a la versión definitiva del programa, estas se muestran en la sección (3.3.4).

3.3.1.1. Funciones de acceso directo a memoria

Para realizar el acceso a la memoria, se implementaron las siguientes funciones:

- **char leeByte(long direccion)**. Esta función recibe una dirección absoluta de la partición, realiza directamente la búsqueda en la memoria, y devuelve el byte encontrado en esa dirección. Es importante mencionar que esta función es la única que originalmente tenía acceso físico al medio, por lo que en cada una de las pruebas esta función se ve sujeta a cambios. El código de esta función en general resulta:

```
unsigned char leeByte(unsigned int direccion)
{
    char temp;
    char *t = &temp;
       fseek(miArch,direccion,SEEK_SET);
       fread((void*)t,1,1,miArch);
       return temp;
}
```

  Sin embargo, la función fseek y fread se ven sustituidos por los comandos de lectura a puerto en el Rabbit (ver 3.3.4)

- **int leeInt(long direccion)**. Esta función realiza una lectura de un entero 'big endian' (2 bytes, donde se lee primero el byte menos significativo), a partir de dos llamadas a leeByte. El código fuente de esta función es:

```
unsigned int leeInt(unsigned int direccion)
{
      return     (unsigned     int)(leeByte(direccion))+(unsigned
int)(256*leeByte(direccion+1));
}
```

- **char* leeString(int direccion, int tamanio)**. Esta función realiza la lectura de varios bytes (en cantidad por el segundo parámetro) almacenándonos en un arreglo de caracteres y devolviendo el apuntador al inicio de la cadena. En la implementación final en el Rabbit fue necesario modificar el contrato de esta función debido a las limitaciones estáticas de la misma. Por lo mismo fue igualmente necesario reducir el tamaño máximo permitido a un valor fijo y compilable en 512K de RAM. El código fuente de esta función se presenta en seguida.

```
unsigned char* leeString(unsigned int direccion, int tamanio){
    int i;
    if(tamanio>2048) tamanio = 2048;
    for(i=0;i<tamanio;i++){
        readTemp[i]=leeByte(direccion+i);
    }
    readTemp[i] = '\0';
    return readTemp;
}
```

- **void inicializa(void)**. Esta función tiene el propósito de realizar toda la inicialización de los parámetros FAT, como es el número de bytes por sector, de sectores por clúster, sectores por FAT, entradas raíz máximas, y por supuesto todas las ecuaciones definidas en la tabla (2.1.2.2). Esta función parte del hecho que leeByte ya funciona, por lo que primero es necesario tener acceso al medio físico de almacenamiento. El código fuente de esta función es el siguiente:

```
void inicializa()
{
      if(leeByte(16)){
            bytesXSector = leeInt(11); //11 y 12
            sectoresXCluster = (unsigned int)leeByte(13);
      sectoresReservados = leeInt(14); //14 y 15
            //16 es number of FAT's
            entradasRaiz = leeInt(17); //17 y 18
            //19 y 20 son para #sectores, 21 para media descr.
            sectoresXFAT = leeInt(22);
            offsetParticion = 0;
            offsetFAT         =         offsetParticion         +
sectoresReservados*bytesXSector;
            offsetRaiz = offsetFAT+2*sectoresXFAT*bytesXSector;
            offsetData = offsetRaiz+entradasRaiz*32;
            dirActual = offsetRaiz;
            curDirArch.esDir = 1;
            strcpy(curDirArch.nombre,"/");
            curDirArch.dirInicio = 0;
            printf("Bytes por sector: %d\n",bytesXSector);
            printf("Sectores             por             cluster:
%d\n",sectoresXCluster);
            printf("Sectores                              reservados:
%d\n",sectoresReservados);
            printf("Entradas raíz máximas: %d\n",entradasRaiz);
            printf("Sectores por FAT: %d\n",sectoresXFAT);
            printf("Offset Directorio Raíz: %lx\n",offsetRaiz);
            printf("Offset FAT1: %lx\n",offsetFAT);
      }
      else{
            exit(1);
      }
}
```

En el caso de la implementación final del Rabbit, se verá en la sección (3.3.4) que debido a que se implementó un caché especial para acelerar el proceso de lectura, y que la inicialización debe ser ejecutada ANTES de activar el caché, se utilizaron funciones especiales, similares a leeByte y leeInt. Además, el cambio de directorio y la inicialización física de la unidad se llevó a cabo en esta misma instrucción.

- **void cd(archivo *arch).** Esta función realiza el proceso de cambio de directorio, al directorio descrito en el apuntador a la estructura correspondiente. Como primera validación, verifica que efectivamente se trate de un directorio, en caso contrario no realiza ninguna acción, y envía a la salida estándar del programa un mensaje de error. Esta función está implementada de la siguiente manera:

```
void cd(archivo *arch)
{
    unsigned long ap,x;
    unsigned long a;
    char fecha[16];
    char tmp[4];
    unsigned int tmp2;
    printf("Llegamos aquí");
      if(arch->esDir>0)
    {
        //Sí es directorio, hacer el cambio
        if(arch->dirInicio==0){
            ap = offsetRaiz;
            dirActual = offsetRaiz;
            curDirArch = *arch;
        }
        else
        {
            ap   =   offsetData   +   (arch->dirInicio-2)*(unsigned
long)bytesXSector*(unsigned long)sectoresXCluster;
            dirActual = ap;
            curDirArch = *arch;
            printf("Como fue ROOTDIR, ap = %d o %lx\n\n",ap,ap);
        }

        for(x=0;x<128;x++)
        {

if((leeByte(ap)==0x00)||((dirActual!=offsetRaiz)&&(ap-
dirActual>=bytesXSector*sectoresXCluster))){
                strcpy(archivos[x].nombre,"\0");
                break;
            }
            if((leeByte(ap)==0xE5)||(leeByte(ap+11)==0x0F)){
                //printf("[@%lx]     Ignorando     1     elemento
eliminado... \n",ap);
                ap+=32;
                x--;
                continue;
            }
            if(leeByte(ap+11)==0x08){
             ap+=32;
               x--;
             continue; //es VOLUME LABEL
            }


            //Entrada directorio
            archivos[x].dirEntry = ap;
```

```
                //Nombre
                leeString(ap,8,archivos[x].nombre);
                strcat(archivos[x].nombre,".");
                leeString(ap+8,3,tmp);
                strcat(archivos[x].nombre,tmp);
                archivos[x].nombre[12] = 0x00;
                //printf("[@%lx]                Item                %d:
%s\n",ap,x,archivos[x].nombre);
                if(leeByte(ap+11)==((unsigned char)16))
                {
                 printf(" (directorio) ");
                    archivos[x].esDir = 1;
                }
                else
                {
                    archivos[x].esDir = 0;
                }
                printf("%s\n",archivos[x].nombre);
                //¿Es directorio?

                //Cluster de inicio
                archivos[x].dirInicio = leeInt(ap+26);

                //Fecha

                ltoa((leeInt(ap+0x18)&0x001F),tmp); //Día
                strcpy(fecha,tmp);
                ltoa((leeInt(ap+0x18)&0x01E0)>>5,tmp); //Mes
                strcat(fecha,"/");
                strcat(fecha,tmp);
                ltoa(((unsigned
int)(leeInt(ap+0x18)&0xFE00))>>9,tmp); //Año
                strcat(fecha,"/");
                strcat(fecha,tmp);
                strcpy(archivos[x].fcreacion,fecha);

                //Tamaño
                archivos[x].tamanio                                =
leeInt(ap+28)+leeInt(ap+30)*0xFFFF;

                //Ver si hay más
                if(leeByte(ap+32)==0)
                {
                    strcpy(archivos[x+1].nombre,"\0");
                    break;
                }
                else
                {
                    //Caso normal
                    ap+=32;
                }
            }
          }
       else printf("Este archivo no es directorio");
}
```

- **archivo* myfopen(archivo *arch, int offsetCluster)**. Este archivo realiza la apertura virtual de un archivo individual. Para abrir un archivo por primera vez se deberá inicializar el parámetro offsetCluster en 0; los valores diferentes a 0 son empleados por mycat y mycopy para sobreponerse a la limitante de estructuras estáticas. Esta función almacena en un buffer la lista de los clústeres en donde se encuentra el archivo, pero no realiza lecturas sobre dichos clústeres. Adicionalmente, asigna a este archivo la marca de abierto, para relacionar la información del clúster con

el archivo; una vez ejecutado este proceso la función regresa un apuntador al mismo archivo que fue invocado. Myfopen no debe ser empleado con directorios, ya que tiene resultados impredecibles por el momento, y no se encuentra vinculado a dir(). Este programa se encuentra implementado de la siguiente forma:

```
archivo* myfopen(archivo *arch, int offsetCluster)
{
    int i;
    printf("\nInicializando apertura de archivo %s",arch-
>nombre);
    if(offsetCluster==0) clustersArchivo[0] = arch->dirInicio;
    else clustersArchivo[0] = offsetCluster;
    printf("\n Clusters del archivo: %d,",clustersArchivo[0]);
    for(i=1;i<256;i++)
    {
        clustersArchivo[i]                                    =
leeInt(offsetFAT+2*clustersArchivo[i-1]);
        if(clustersArchivo[i]>=0xFFF8) break;
    }
    printf("\n");
    arch->abierto = 1;
    return arch;
}
```

El algoritmo general para obtener los clústeres de cualquier archivo se menciona a continuación.

1. Inicio: Recibir como parámetro la estructura del archivo.
2. Obtener la dirección del clúster de inicio del archivo.
3. Almacenar número de clúster en estructura de clusterArchivo.
4. Buscar el clúster en la tabla FAT (Ecuación FAT-Cluster N).
5. Leer datos del FAT-cluster.
6. Si el cluster es terminador (se lee dato >FFF7h), ir a paso 9.
7. En caso contrario, cambiar cluster activo = número de cluster leído.
8. Ir a paso 3.
9. Regresar estructura clusterArchivo.
10. Fin.

Importante: En teoría, si una partición se encuentra perfectamente defragmentada, todos los clusters de un archivo serán consecutivos. En la práctica, es altamente probable que esto ocurra. Por ello, puede resultar tentador el tratar de leer el tamaño completo del archivo a partir del offset Cluster-Inicio, y en algunos casos funcionará. Sin embargo, resulta indispensable el ejecutar este algoritmo cada vez que se desee abrir un archivo, para tener un sistema que funcione aún en el peor de los casos.

3.3.1.2. Estructuras estáticas

- **typedef struct archv{...} archivo**. Esta estructura es la base de las funciones comunes, y provee una forma de comunicación entre las distintas funciones. El tipo de datos archv contiene lo siguiente:

```
typedef struct archv{
      char nombre[13];
   short esDir;
```

```
        unsigned long tamanio;
        char fcreacion[11];
        unsigned long dirInicio;
        unsigned long dirEntry;
        short abierto;
    } archivo;
```

Esta estructura se utiliza para crear el arreglo **archivo archivos[128]**, que es la estructura estática empleada para almacenar los elementos de directorio. Esta estructura es regenerada cada vez que se invoca el comando cd(archivo *arch).

- **int clustersArchivo[256]**. Esta sencilla estructura constituye el buffer de apertura de todo archivo, y se emplea para registrar los clústeres del archivo abierto. Por lo mismo solamente se puede tener un archivo abierto a la vez.

3.3.2. Emulación de FAT en consola

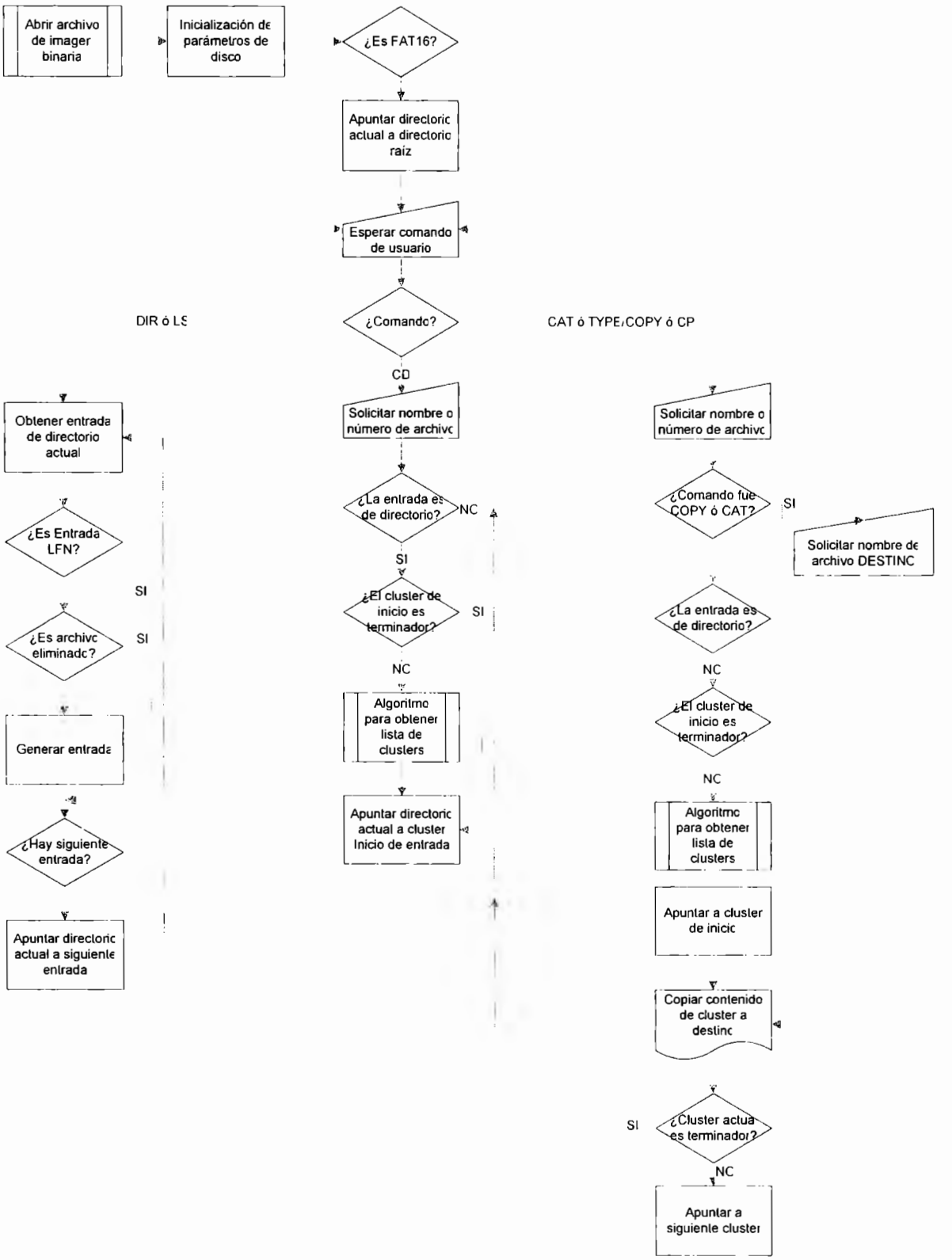


Figura 3.3.2.1. Algoritmo general del emulador en consola.

El emulador en consola fue un programa de prueba elaborado con el propósito de evaluar la validez de todas las funciones y algoritmos de FAT desarrollados; por lo mismo utiliza todas las funciones anteriormente descritas. La interacción con el usuario se lleva a cabo mediante el uso de comandos tecleados, en forma análoga a un símbolo de comandos (o shell) de un sistema operativo. La implementación soporta 3 comandos diferentes:

- **DIR o LS:** Esta función enlista todos los archivos registrados en el buffer archivos[128] descrito en la sección anterior; para poderse emplear hay que haber invocado previamente el comando cd(*archivo), de lo contrario se tendrán resultados inesperados. Este comando no requiere de parámetros adicionales en consola. Este comando se implementó mediante la función **void dir()**, que contiene lo siguiente:

```
void dir()
{
    archivo *arch;
    int x;
    arch = &curDirArch;

    printf("Leyendo directorio %s\n",arch->nombre);
        //Despliegue
    for(x=0;x<128;x++)
    {
        if(archivos[x].nombre[0]=='\0') break;
        printf("[0x%x] ",archivos[x].dirEntry);
        printf("%d)    %s    %s    %d    bytes            (cluster
%d)\n",x,archivos[x].nombre,((archivos[x].esDir==1)?"   <DIR>   ":"
"),archivos[x].tamanio,archivos[x].dirInicio);
    }


}
```

- **CD.** Este comando permite el cambio de directorio. Al ser invocada recibe como parámetros el índice del directorio deseado, el cual se muestra al llamar el comando DIR o LS. Este comando se implementó mediante la función **void cd(archivo *arch)**, el cual se describe en la sección precedente.
- **COPY/CP y CAT/TYPE**. Estas funciones son dos enfoques de la misma utilidad: mientras que CAT muestra el contenido de cualquier archivo en pantalla, COPY permite copiar el "archivo virtual" a un nuevo archivo real del sistema donde se ejecuta el emulador. Ambos comandos requieren previamente invocar el comando myfopen, que devolverá la lista de los primeros 256 clústeres donde se encuentra el archivo. En caso de que el archivo utilice más de 256 clústeres, es responsabilidad de ambas funciones el volver a llamar myfopen con el offsetCluster en 256. El comando COPY se implementó mediante la función **unsigned int mycopy(archivo *arch, char *destino)**, mientras que CAT se implementó con **unsigned int mycat(archivo *arch)**.

```
unsigned int mycopy(archivo *arch,char *destino)
{
    int x,i,ap,ncl;
    long bytesLeidos;
    FILE *archW32;
    x=0;
```

```
    if(arch->abierto==1)
    {
        bytesLeidos = 0;
        archW32 = fopen(destino,"wb");
        ncl = arch->tamanio / bytesXSector*sectoresXCluster;
        if(archW32==NULL) return 1;
        do
        {
            ap          =          offsetData+(clustersArchivo[x]-
2)*bytesXSector*sectoresXCluster;
            if(ncl==x)
            {
                fwrite(leeString(ap,arch->tamanio-
bytesLeidos),arch->tamanio-bytesLeidos,1,archW32);
                break;
            }
            else
fwrite(leeString(ap,bytesXSector*sectoresXCluster),bytesXSector*s
ectoresXCluster,1,archW32);
          x++;
          if(x==255)
          {
              myfopen(arch,clustersArchivo[x]);
              x = 0;
          }
        } while(clustersArchivo[x]<0xFFF8);

    }
}
unsigned int mycat(archivo *arch)
{
    int x,i,ap,ncl;
    long bytesLeidos;
    x=0;
    if(arch->abierto==1)
    {
        bytesLeidos = 0;
        ncl = arch->tamanio / bytesXSector*sectoresXCluster;
        do
        {
            ap          =          offsetData+(clustersArchivo[x]-
2)*bytesXSector*sectoresXCluster;
            //printf("Para  cluster  %d  inicializo  apuntador  en
%d",clustersArchivo[x],ap);
            if(ncl==x)
            {
                printf("%s",leeString(ap,arch->tamanio-
bytesLeidos));
                break;
            }
            else
            {

printf("%s",leeString(ap,bytesXSector*sectoresXCluster));
            }
          x++;
          if(x==255)
          {
              myfopen(arch,clustersArchivo[x]);
              x = 0;
          }
        } while(clustersArchivo[x]<0xFFF8);

    }
}
```

Para mayor información sobre este emulador, consulte los anexos donde podrá encontrar el código fuente completo.

3.3.3. Emulación de FAT en servidor web

El emulador FAT para servidor web consta de los mismos elementos que el emulador en (3.3.2). Sin embargo, tiene 2 diferencias fundamentales: el llamado de las funciones no es por espera al usuario, sino que el programa se inicia con todos los parámetros, ejecuta lo necesario, genera la salida adecuada, y termina con un código de salida 0; en segundo lugar, todas las salidas deberán estar formateadas en formato HTML para poder desplegar adecuadamente los resultados. Si desea mayor información sobre este emulador, consulte los anexos donde podrá encontrar el código completo.

3.3.4. Programa integrado para el Rabbit

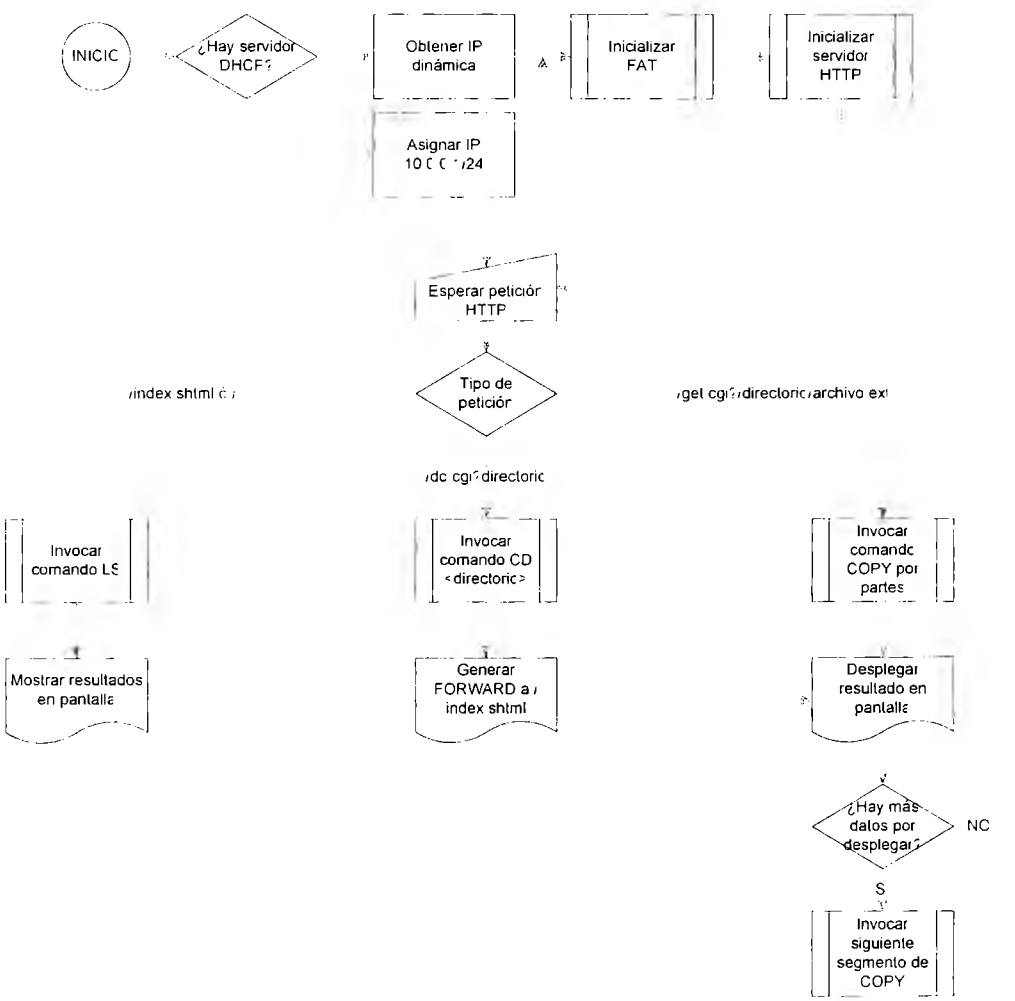


Imagen 3.3.4.1. Algoritmo general del servidor web

El programa integrado para el Rabbit utiliza en su mayoría las funciones previamente descritas. Sin embargo por su naturaleza, se generaron algunas funciones adicionales para los procesos con el servidor web del Rabbit, y además se modificaron algunas de las funciones base para adecuarlo a las limitaciones de velocidad de comunicación entre el Rabbit y la memoria Flash.

3.3.4.1. Lectura en caché

Al realizar las primeras pruebas en el Rabbit teniendo plenamente integrada la interpretación y la memoria Flash, se observó que los tiempos de ejecución e inicialización eran demasiado grandes (lo que en PC se inicializaba en 2 segundos, en el Rabbit tardó 25 minutos). Por lo mismo se decidió implementar una lectura en caché empleando lecturas por bloques completos a la memoria Flash, reduciendo el tiempo de intercambio de mensajes y aumentando considerablemente la velocidad de lectura.

El procedimiento para la lectura en caché se basa en leer un bloque fijo (o clúster absoluto) de 256 bytes para la primera lectura de byte. Si el siguiente byte pertenece al mismo bloque, entonces la lectura ya no requiere de acceso a la tarjeta Flash. En caso de requerir una lectura a otro bloque, se solicita el bloque completo. En las pruebas realizadas, donde un bloque para una memoria de 128 MB es ½ cluster, se observó que el tiempo de espera disminuyó de un promedio de 5 minutos a un promedio de 5 segundos (es decir, prácticamente 60 veces más rápido). Por su lado, en el caso de la computadora, no se obtuvo un tiempo mejor de lectura, lo cual se explica por el hecho de que Windows automáticamente maneja la apertura de archivos utilizando páginas de memoria de 2KB, es decir que lo que se lee del disco en realidad se está leyendo a través de un caché con DMA asíncrono.

Para la lectura en caché se realizaron los siguientes pasos:

1. Se renombraron las funciones leeByte, leeInt y leeString por leeByteSC, leeIntSC y leeStringSC (SC = Sin Caché), y se modificaron todas las referencias entre ellas. Estas funciones se emplean en la secuencia de inicialización, debido a que en esos casos sí es necesario tener lecturas y escrituras byte por byte.
2. Se creó una función leeCluster, que implementa la lectura a un clúster absoluto.
3. Se modificó la función leeByte de tal forma que a su vez dependa de leeCluster para tener acceso al medio, y que solamente solicite datos en bloque a la memoria y cuando en RAM.

A continuación se detallan las funciones generadas:

- **void leeCluster(unsigned long numClusterAbs).** Esta función lee un clúster absoluto (definiendo un clúster absoluto como un espacio de 256 bytes donde el clúster absoluto 0 es el inicio de la memoria Flash), y la escribe en una variable global llamada **bloqueAbsN**, la cual es leída posteriormente por leeByte. leeCluster no realiza ninguna acción de inteligencia, únicamente solicita a la memoria Flash un bloque de 256 bytes de longitud.

```
void leeCluster(unsigned long numClusterAbs)
```

```
{
    unsigned char temp[6];
    unsigned long cosa;
    int c,d;
    unsigned long direccion;
    direccion = numClusterAbs*maxBuffAdd;
    cosa = 256;
    temp[0] = 0x90;
    temp[1] = (direccion)%cosa;
    temp[2] = (direccion/(cosa))%cosa;
    temp[3] = (direccion/(cosa*cosa))%cosa;
    temp[4] = (direccion/(cosa*cosa*cosa))%cosa;
    temp[5] = 0x01;
    for(c=0;c<6;c++) serDputc(temp[c]);
    bloqueAbsN = numClusterAbs;
    for(c=0;c<maxBuffAdd;c++)
    {
      d = -1;
      while(d == -1){
         d = serDgetc();
      }
       bloqueAbs[c] = (unsigned char)d;
      //printf("%d)%x   ",c,bloqueAbs[c]);
    }
    return;
}
```

- **unsigned char leeByte(unsigned long direccion)**. Esta función modificada, compara el clúster absoluto al que pertenece el byte solicitado contra el clúster absoluto actualmente en el caché. Si no hay bloque en caché, o bien si no son iguales, invoca a la función leeCluster para obtener el bloque completo, y posteriormente devuelve el byte solicitado. En caso de pertenecer al mismo bloque, únicamente devuelve el byte solicitado.

```
unsigned char leeByte(unsigned long direccion)
{
      //printf("Direccion      %lx      @      bloque      %d.
",direccion,direccion/maxBuffAdd);

      if(direccion/maxBuffAdd==bloqueAbsN){
      //printf("Actual = %d leo offset %d",bloqueAbsN,(direccion-
bloqueAbsN*maxBuffAdd));
      return bloqueAbs[(direccion-bloqueAbsN*maxBuffAdd)];
   }
   else
   {
      leeCluster(direccion/maxBuffAdd);
      //printf("AHORA     Actual     =     %d     leo     offset
%d",bloqueAbsN,(direccion-bloqueAbsN*maxBuffAdd));
      return bloqueAbs[(direccion-bloqueAbsN*maxBuffAdd)];
   }
}
```

3.3.4.2. Funciones de procesos web

Con el fin de acoplar las funciones previamente generadas con las funciones propias del Rabbit, se realizaron las siguientes funciones que reciben las peticiones del usuario.

- **int changeDir(HttpState *estado)**. Esta función es invocada por el Rabbit cuando el usuario hace clic en un hipervínculo de un directorio en el API web. La estructura HttpState es parte del contrato de las funciones invocadas directamente por la

biblioteca http.lib de Rabbit, por lo que no se puede modificar. Esta función realiza los cambios de directorio gráficos, para después mandar llamar el comando cd previamente descrito.

```
int changeDir(HttpState* estado)
{
   int cdIndex;
   printf("Llegué a inicio de changeDir\n");
      strcpy(stateURL,(estado->url+8));
   cdIndex = atoi(stateURL);
   printf("Obtuve cdIndex = %d\n",cdIndex);
   strcpy(curDir,archivos[cdIndex].nombre);
      printf("Cambié directorio actual a %s\n",curDir);
   printf("StateURL = %s\n",stateURL);
   printf("redirectTo = %s\n",redirectTo);
   cd(&archivos[cdIndex]);
   getCurDir();
   cgi_redirectto(estado,redirectTo);
      return 0;
}
```

- **void getCurDir()**. Esta función es una versión modificada de DIR/LS, pero con su salida adaptada para el servidor web de Rabbit. Esta salida consiste en copiar toda la salida de los datos del directorio en HTML a una variable accesible desde el SHTML que se despliega al usuario final. Esta función no hace uso del comando DIR, ya que la implementación es diferente. La implementación de esta función se hizo de la siguiente forma:

```
void getCurDir()
{
   int x,y;
   char siz[10];
   char id[3];
   char *ptrDirText;
   ptrDirText = dirText;
   //Ahora sí viene lo bueno
   strcpy(ptrDirText," ");
   for(x=0;x<(sizeof(archivos)/sizeof(archivos[0]));x++)
   {
       if(archivos[x].nombre[0]==0x00){

       break;
       }
       printf("entradas[%d]\n",x);
      printf("\n.nombre = %s",archivos[x].nombre);
      printf("\n.esDir = %d",archivos[x].esDir);
      printf("\n.fcreacion = %s",archivos[x].fcreacion);
      printf("\n.tamanio = %d",archivos[x].tamanio);
      utoa(archivos[x].tamanio,siz);

       if(archivos[x].esDir==1)
      {

         //Es directorio
         strcpy(tmp,"");
         strcpy(tmp,"<tr><td>(DIR)</td><td><a href='do.cgi?");
         utoa(x,id);
         strcat(tmp,id);
         strcat(tmp,"'>");
         strcat(tmp,archivos[x].nombre);
         strcat(tmp,"</a>\n</td><td>");
         strcat(tmp,archivos[x].fcreacion);
         strcat(tmp,"</td><td></td></tr>\n");
```

```
            }
            else
            {
             //No es directorio
                strcpy(tmp,"<tr><td></td><td>");
                strcat(tmp,archivos[x].nombre);
                strcat(tmp,"</td><td>");
                strcat(tmp,archivos[x].fcreacion);
                strcat(tmp,"<td>");
                strcat(tmp,siz);
                strcat(tmp,"</td></tr>\n");
            }
            strcat(dirText,tmp);
                printf("strlen(tmp) = %d\n", strlen(tmp));
                for(y=0;y<strlen(tmp);y++){
                   printf("%d.%c ",y,tmp[y]);
                }
        }
        printf("\n");
        for(x=0;x<strlen(dirText);x++){
            printf("%c",dirText[x]);
        }
    }
```

3.3.5. Emulador de memoria serial

Debido a las complicaciones presentadas al tratar de integrar el Rabbit con la tarjeta Secure Digital vía SPI (ver secciones de Resultados y de Problemas), se decidió implementar una interfase de prueba entre la computadora y el Rabbit a través de una interfase USB, constituyendo así un emulador de memoria serial.

El emulador en memoria serial es un programa desarrollado en C#.NET, que básicamente consiste en una máquina de estados en función de lecturas y escrituras al chip FTDI utilizando programas intermedios, conectado al puerto USB de la computadora, el cuál es el núcleo del USB232M. Posee una arquitectura basada en hilos independientes de lectura, escritura, inicialización y despliegue, cada uno interconectado con los demás a través del uso de semáforos booleanos. El emulador consta de 3 partes: el driver de bajo nivel de FTDI, el DLL intermedio que permite la comunicación libre de apuntadores entre C#.NET y el driver, y el programa propiamente en .NET. La documentación del driver de bajo nivel queda fuera de los objetivos del presente documento, por lo que para mayor información se deberá remitir a [18].

3.3.5.1. DLL Intermedio

Este DLL fue desarrollado inicialmente por dos de los integrantes del equipo (Ludwig Méndez y Rafael Tello) durante un proyecto en semestres anteriores, y algunas de las funciones externas que contiene son las siguientes:

- **unsigned int FT_Open()**. Esta función abre la conexión al dispositivo USB más cercano. Parte del hecho de que existe sólo un chip FTDI conectado al sistema, de lo contrario sería necesario implementar un FT_Open(int numeroDeChip). La llamada a bajo nivel manda llamar al dispositivo FTDI 0.

- **unsigned int FT_Close()**. Esta función cierra la conexión al dispositivo USB.
- **unsigned int FT_SetBaudRate(unsigned int bitRate)**. Esta función permite
- **unsigned int FT_Read(UCHAR *dato, unsigned long tamanio).** Esta función permite realizar la lectura de del buffer de entrada del USB. Un punto importante de esta función, a diferencia de las funciones de lectura de puerto o por interrupción, es que no regresa hasta que efectivamente se recibe un byte. Esto puede resultar positivo por un lado, ya que una instrucción garantiza la lectura; sin embargo, es necesario ejecutar el comando en un thread independiente, desactivarlo por defecto, y cada segundo (o el tiempo requerido), levantar el thread para revisar si ya hay un dato nuevo. De lo contrario puede causar el congelamiento del programa que lo invoca.
- **unsigned int FT_Write(UCHAR *dato, unsigned long tamanio)**. Esta función escribe <tamanio> bytes apuntados por dato en el buffer de salida de USB. A diferencia de FT_Read, este comando termina en cuanto se envía al buffer de salida, no en cuanto se envíe al otro dispositivo.
- **unsigned int FT_PurgeTX() y unsigned int FT_PurgeRX()**. Estas funciones limpian respectivamente el buffer de salida y de entrada del USB. Estas funciones resultan particularmente útiles cuando no se tiene certeza de que el dispositivo al otro lado no haya transmitido basura mientras se atendía algún otro elemento del búfer, como por ejemplo si se tiene conectado un ADC. También sirve en caso de perder sincronía debido a un error de transmisión.

3.3.5.2. Emulador de Secure Digital

El emulador de Secure Digital no es otra cosa que una máquina de estados programada en C#.NET, que espera un determinado comando completo (de 6 bytes). Al recibirlo, dependiendo del contenido del primer byte toma determinadas acciones. El proceso de lectura del puerto se mantiene en un hilo independiente, al igual que el proceso de escritura. El programa tiene implementadas, entre otras, las siguientes funciones:

- **public void escuchaUSB()**. Esta función se encarga de esperar cualquier dato enviado al USB. Esta función se encuentra controlada utilizando semáforos, por lo que se inicia siempre y cuando se haya inicializado USB adecuadamente, no exista algún proceso de escritura, y se reactiva a sí misma cada milisegundo.
- **public void transmiteAUSB().** Esta función realiza la transmisión al puerto del número de bytes en el buffer del programa de salida. Esta función se encuentra controlada por semáforos, y se inicia al existir una petición de escritura por parte de alguna función del sistema, al no haber ningún proceso de lectura pendiente, y al encontrarse adecuadamente inicializado el puerto USB.
- **public void reintentarConnect().** Esta función se utiliza cuando el programa se inicia y no hay ningún chip FTDI conectado. Su propósito es reintentar cada segundo

conectarse al dispositivo; en caso de lograrse, activa el semáforo de USB inicializado y se desactiva a sí mismo. De igual forma, al haber una desconexión del dispositivo, este programa se reactiva cada segundo hasta reconectarse de nuevo.

- **public string leeString(long direccionOffset, int tamanio).** Esta función es la implementación en C# de la función leeString(char* buffer, long dirOffset, int tamanio) desarrollada en los emuladores de FAT. Esta función accede a nivel binario al sistema de archivos, obtiene el número de bytes requeridos, y devuelve por referencia la cadena generada.

3.4. Hardware

El desarrollo en hardware se realizó en la misma forma como se indicó en el marco teórico, con la única excepción la conexión del emulador de la tarjeta Flash utilizando el DLP232M de DLP Design.

3.4.1. Conexión Rabbit – PC vía USB (para emulador de memoria serial)

El dispositivo empleado para lograr la interconexión es el DLP-USB232M, de DLP Design, que utiliza el chip de FTDI (uno de los fabricantes de chips controladores para USB más importantes en el mercado). Este dispositivo permite una separación física y lógica de la complejidad de USB en el dispositivo, y opcionalmente en la computadora. Es decir, para el Rabbit la operación del USB es transparente, ya que se conecta al dispositivo como si se tratara de un módem. Para la PC la operación *puede ser* transparente, dependiendo del driver utilizado: puede instalarse un emulador de puerto serial (que se instala como COM4 o COM5), utilizable en cualquier aplicación como HyperTerminal, o bien en C o C# como un puerto más de la computadora; por el otro lado, se puede instalar el controlador tradicional FIFO, que permite un grado más complejo de configuración, velocidades mucho mayores (los integrantes del equipo han probado más de 1Mbps con un DSP), y un control total del proceso de transmisión.

En cualquier caso, el diagrama de conexión del USB232M con el Rabbit es el siguiente:

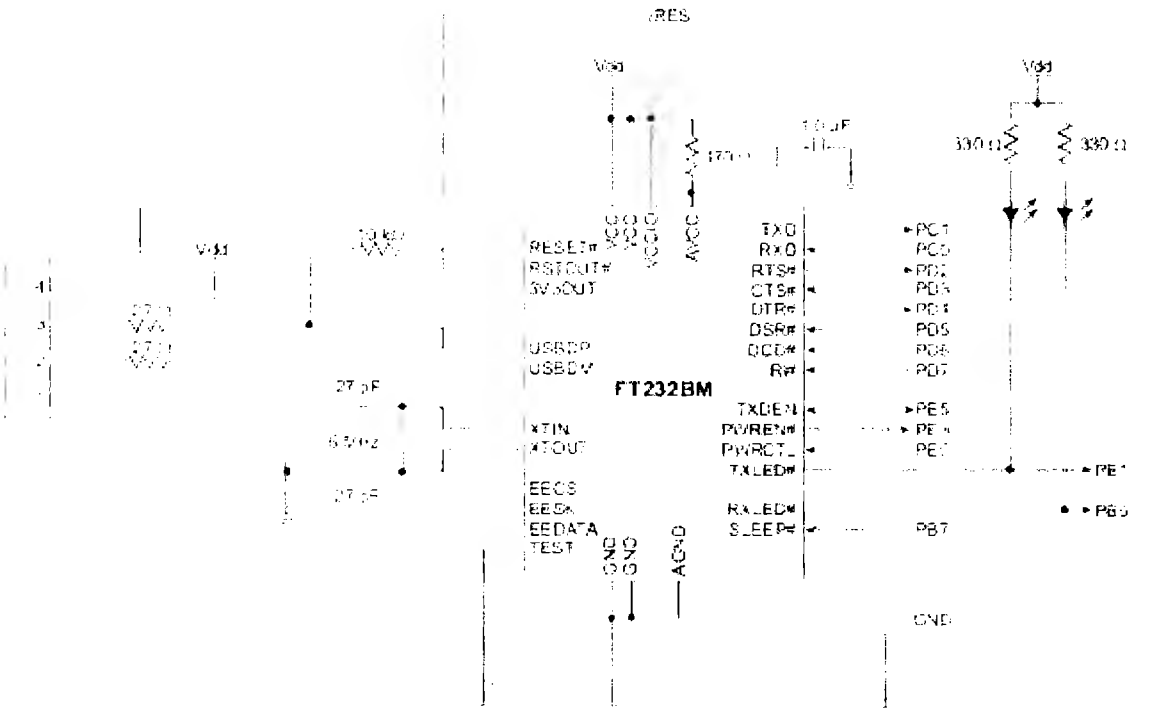


Figura 3.4.1.1. Diagrama de conexión entre el USB232M y el Rabbit

Si bien el diagrama anterior se refiere al puro chip de FTDI, existe un mapeo total entre los pines del chip FTDI y los pines del DLP-USB232M, lo cual se comprueba fácilmente mirando el encapsulado completo.

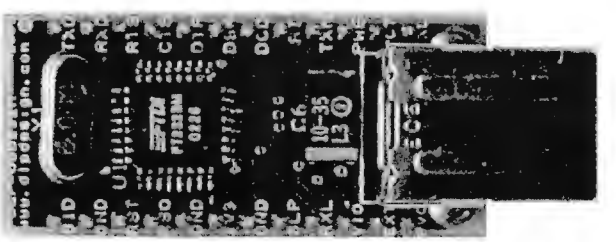


Figura 3.4.1.2. Vista superior del DLP-USB232M

4. Presentación y análisis de resultados

4.1. Emuladores en PC

4.1.1. Emulador de FAT en PC

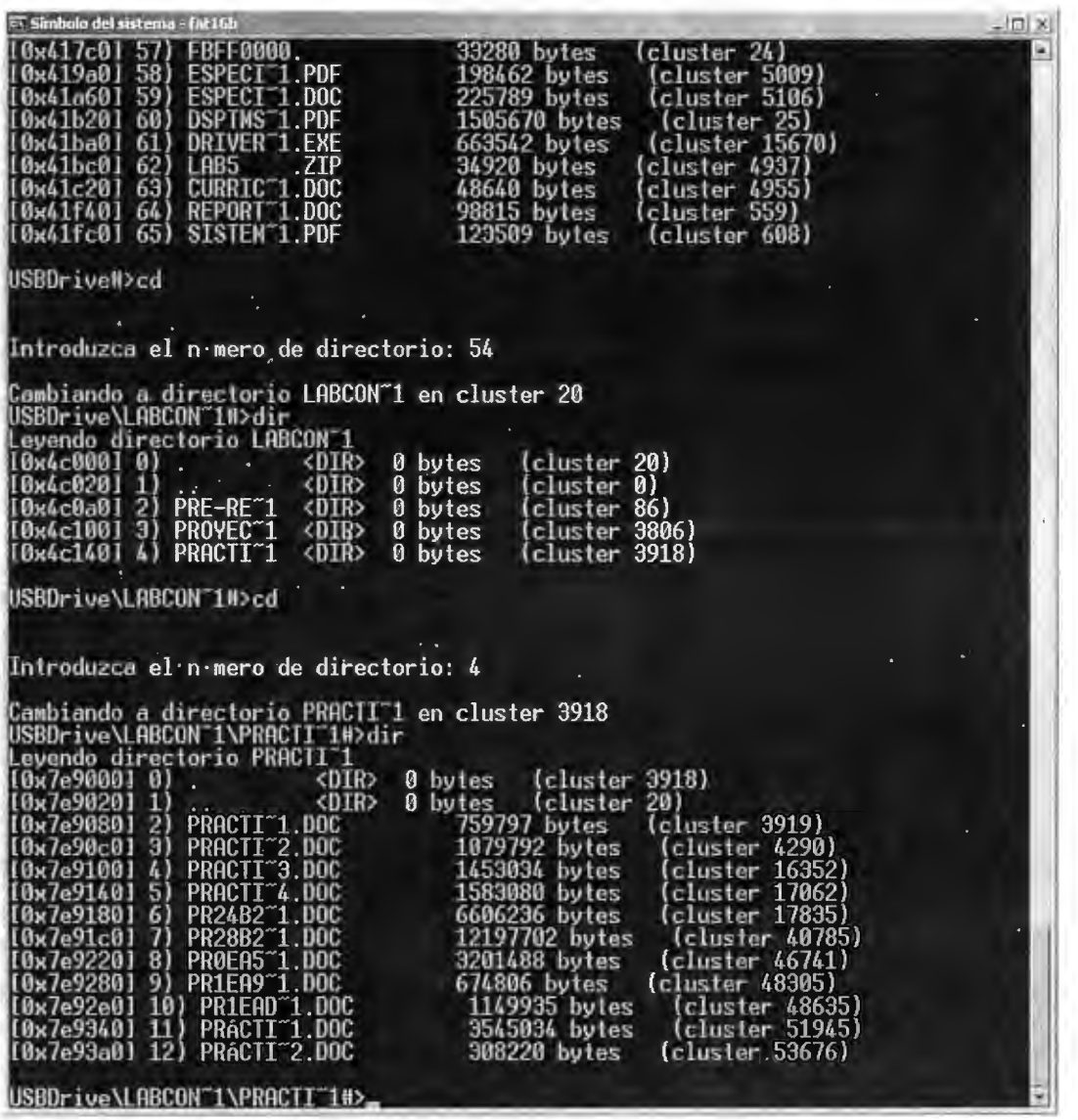


Figura 4.1.1.1. Muestra del programa FAT16b.exe en funcionamiento

4.1.1.1. Volcado de resultados de una prueba

```
C:\Documents and Settings\Rafael Tello\Mis documentos\flash2web\fat16b
MicroKernel  .1.

flash2web. Proyecto de IngenierÝa
Concepción: Ludwig Méndez y Rafael Tello

Inicializando imagen binaria de disco...
```

```
                                < OK >
Inicializando servicio FAT16...
Bytes por sector: 512
Sectores por cluster: 4
Sectores reservados: 8
Entradas raÝz mßximas: 512
Sectores por FAT: 248

                                < OK >
Montando directorio raiz...     < OK >

Inicializando prompt...         < OK >

USBDrive#>dir <--- //tecleado por el usuario
Leyendo directorio /
[0x3f4e0] 0) TEC200~1  <DIR>  0 bytes   (cluster 2)
[0x3f520] 1) NESTLÉ    <DIR>  0 bytes   (cluster 3)
[0x3f560] 2) PERSON~1  <DIR>  0 bytes   (cluster 4)
[0x3f600] 3) CONCEP~1  <DIR>  0 bytes   (cluster 5)
[0x3f640] 4) MISIMÁ~1  <DIR>  0 bytes   (cluster 6)
[0x3f680] 5) TEMPOR~1  <DIR>  0 bytes   (cluster 7)
[0x3f760] 6) ACUERD~1.DOC         83967 bytes   (cluster 28)
[0x3f7c0] 7) COEVAL~1.XLS         33792 bytes   (cluster 69)
[0x3fac0] 8) ESTADO~1.DOC         39936 bytes   (cluster 27236)
[0x3fd80] 9) INTERO~1.DLL         196605 bytes   (cluster 40661)
[0x3fde0] 10) EXCELP~1.PDB         28160 bytes   (cluster 27041)
[0x3fe40] 11) EXCELP~1.EXE         40960 bytes   (cluster 27301)
[0x3fea0] 12) INTERO~2.DLL         57344 bytes   (cluster 40757)
[0x40160] 13) CIRAS2~1.PDF         245051 bytes   (cluster 23851)
[0x401a0] 14) VISIOLI .PDF         342395 bytes   (cluster 24745)
[0x401e0] 15) MOTORC~1  <DIR>  0 bytes   (cluster 24913)
[0x40220] 16) RESPALDO.ACE         1200545 bytes   (cluster 25059)
[0x402e0] 17) MAMBOY~1.DOC         136702 bytes   (cluster 25646)
[0x40340] 18) CLASEA~1.ASM         2709 bytes   (cluster 25663)
[0x40380] 19) CLINICA .MPP         219645 bytes   (cluster 49618)
[0x40480] 20) P8      .ZIP         3697 bytes   (cluster 5126)
[0x404a0] 21) P8        <DIR>  0 bytes   (cluster 5128)
[0x40580] 22) CANADA~1.PDF         303071 bytes   (cluster 237)
[0x405e0] 23) CANADA~2.PDF         85535 bytes   (cluster 385)
[0x40640] 24) CANADA~3.PDF         38217 bytes   (cluster 427)
[0x406a0] 25) CANADA~1.DOC         140798 bytes   (cluster 446)
[0x406c0] 26) BOOTEX  .LOG         1522 bytes   (cluster 8)
[0x40740] 27) EJEMPLOS.DOC         41472 bytes   (cluster 515)
[0x40780] 28) PN!     .DOC         47104 bytes   (cluster 536)
[0x40800] 29) VISION  .DOC         19968 bytes   (cluster 1044)
[0x40860] 30) EXURBA~1.DOC         72703 bytes   (cluster 1054)
[0x40940] 31) CAMION~1.GIF         829 bytes   (cluster 9)
[0x40980] 32) SAECFI~1.ACE         662024 bytes   (cluster 1090)
[0x409a0] 33) MAIN    .CPP         3627 bytes   (cluster 10)
[0x40a00] 34) PORTTA~1.H           1952 bytes   (cluster 12)
[0x40a20] 35) PUERTO  .H           10139 bytes   (cluster 13)
[0x40ac0] 36) PRIMER~1.DOC         105471 bytes   (cluster 1414)
[0x40b40] 37) PORTAD~1.DOC         149502 bytes   (cluster 1466)
[0x40ba0] 38) DOCUME~1.DOC         350203 bytes   (cluster 1539)
[0x40c00] 39) PORTAD~2.DOC         85503 bytes   (cluster 1710)
[0x40c40] 40) OPCION~1.JPG         23427 bytes   (cluster 1752)
[0x40ca0] 41) OPCION~2.JPG         27255 bytes   (cluster 1764)
[0x40ce0] 42) OPCION~3.JPG         23096 bytes   (cluster 1778)
[0x40d20] 43) OPCION~4.JPG         17302 bytes   (cluster 1790)
[0x40d80] 44) NIS&KE~1.PPT         245245 bytes   (cluster 1799)
[0x40e20] 45) INFLUE~1  <DIR>  0 bytes   (cluster 18)
[0x40ee0] 46) FXSASSER.EXE         151694 bytes   (cluster 2090)
[0x40f20] 47) FXSASSER.LOG         120 bytes   (cluster 19)
[0x40fe0] 48) PRE-RE~1.DOC         675830 bytes   (cluster 2165)
[0x41040] 49) PREREP~1.DOC         108031 bytes   (cluster 2495)
[0x410a0] 50) PRE-RE~2.DOC         55296 bytes   (cluster 2548)
[0x41100] 51) PRE-RE~3.DOC         175614 bytes   (cluster 2575)
[0x41140] 52) SHARPLAB.ZIP         120319 bytes   (cluster 2661)
[0x411a0] 53) PRACTI~1.DOC         759797 bytes   (cluster 2720)
[0x411e0] 54) LABCON~1  <DIR>  0 bytes   (cluster 20)
[0x416c0] 55) ACUERD~2.DOC         176638 bytes   (cluster 4818)
[0x417a0] 56) MASTER  .XLS         29696 bytes   (cluster 199)
[0x417c0] 57) FBFF0000.            33280 bytes   (cluster 24)
[0x419a0] 58) ESPECI~1.PDF         198462 bytes   (cluster 5009)
[0x41a60] 59) ESPECI~1.DOC         225789 bytes   (cluster 5106)
[0x41b20] 60) DSPTMS~1.PDF         1505670 bytes   (cluster 25)
```

```
[0x41ba0] 61) DRIVER~1.EXE         663542 bytes   (cluster 15670)
[0x41bc0] 62) LAB5    .ZIP         34920 bytes   (cluster 4937)
[0x41c20] 63) CURRIC~1.DOC         48640 bytes   (cluster 4955)
[0x41f40] 64) REPORT~1.DOC         98815 bytes   (cluster 559)
[0x41fc0] 65) SISTEM~1.PDF         123509 bytes   (cluster 608)
USBDrive#>cd <--- //tecleado por el usuario

Introduzca el n•mero de directorio: 0 <--- //tecleado por el usuario

Cambiando a directorio TEC200~1 en cluster 2
USBDrive\TEC200~1#>dir <--- //tecleado por el usuario
Leyendo directorio TEC200~1
[0x43000] 0) .          <DIR>  0 bytes   (cluster 2)
[0x43020] 1) ..         <DIR>  0 bytes   (cluster 0)
[0x43060] 2) RBLASCO .DOC          482809 bytes   (cluster 27790)
[0x43140] 3) PRACTI~1.DOC          1583592 bytes   (cluster 23971)
[0x436c0] 4) CLINIC~1.PPT          408570 bytes   (cluster 26778)

USBDrive\TEC200~1#>cd <--- //tecleado por el usuario
Introduzca el n•mero de directorio: 1 <--- //tecleado por el usuario
Cambiando a directorio .. en cluster 0
USBDrive#>cat


Introduzca el n•mero de archivo: 18

Inicializando apertura de archivo CLASEA~1.ASM
 Clusters del archivo: 25563,25664, 65535
.INCLUDE "8515def.inc"
.CSEG
.ORG 000
        RJMP    MAIN
.ORG 001
        RJMP    NOAGUA
.ORG 007
        RJMP    CICLOCUENTA
        RET
        RET
        RET
        RET
        RJMP    ANALOGAGUA
.
```

(se omitieron 62 líneas de salida)

```
ANALOGAGUA:
        LDI     R18,$01
        LDI     R16,$02
        OUT     TIMSK,R16
        RETI
USBDrive#>
```

4.1.2. Emulador de FAT en Web

A partir del programa para emular FAT en la PC, se desarrolló un entorno integrado de prueba en la computadora, pero con despliegue de resultados en HTML. Al no contar con las ventajas de las bibliotecas del Rabbit, se tuvo que implementar cada función en forma manual. En forma particular, aspectos como el paso de parámetros del navegador web al programa en C, resultó en primera instancia un problema de diseño, el cual posteriormente, y debido a los graves problemas de seguridad de Internet Information Services (el servidor web de Windows 2000 y Windows XP), no implicó mayor problema después de algunas pruebas.

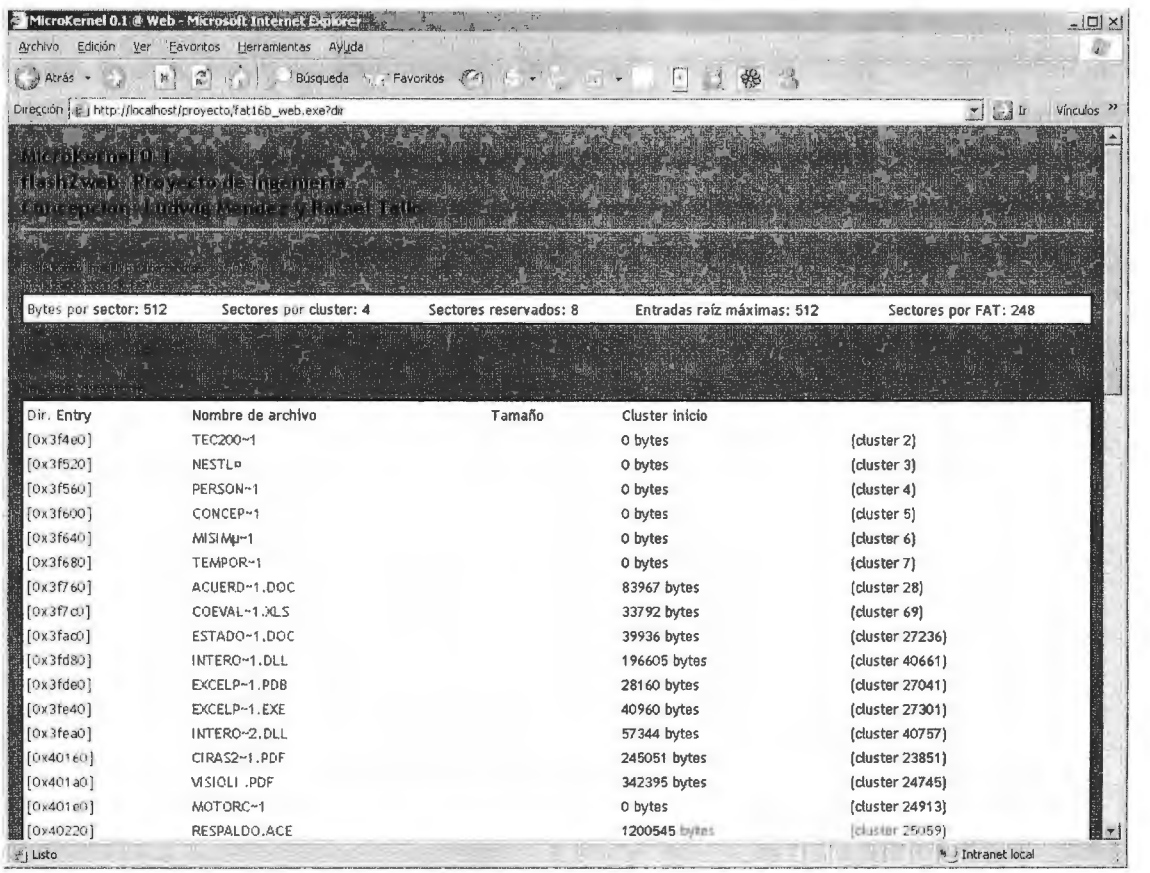


Figura 4.1.2.1. Visualización del programa en ejecución (en carpeta raíz)

En este entorno se incluyeron varias consideraciones que previamente no se habían desarrollado, esto debido a que no se habían podido resolver problemas más básicos. Algunas de estas consideraciones son:

- Se tuvo que modificar enteramente el código de tal forma que a una entrada dada por la barra de direcciones, se tenga una salida definida, y un exit(0) que devuelva el control a IIS. Cualquier dato que el programa solicite después de ser invocado, tiene como consecuencia que se congele el servicio http de Windows. Esto implicó el inconveniente de no tener estado de sesión, como ocurre en el Rabbit; cada vez que se llama el programa se tiene que inicializar todo, y no hay ningún objeto de sesión.
- El generador de contenido HTML tiene completa visibilidad sobre la naturaleza de los archivos (si es directorio, si es un archivo de texto o binario), y en función a eso determinar el link de destino.
- El manejador de archivos individuales reconoce algunos tipos básicos de archivos de texto (HTML, TXT, ASM). Estos tipos de archivo son mostrados en formato amigable, dentro del mismo entorno que el árbol de directorios.
- Los archivos no reconocidos, hacen uso de la función copy() a una carpeta temporal del sistema, donde el usuario IUSR_NOMBRE_DE_EQUIPO tenga permisos de lectura y escritura, pero no de ejecución. Hecho esto, el manejador de archivos hace un Http

Redirect (respuesta 302) a la ubicación del archivo. Con ello, se fuerza a que Internet Explorer o Firefox, quien sin duda tiene una base de datos de tipos MIME mucho más extensa, tome la decisión sobre qué hacer con el archivo.

- En cuanto a apariencia visual, se trabajó por hacer más invisible al usuario la arquitectura interna del programa (por ejemplo, en vez de mostrar el índice del elemento archivos[] en pantalla, forma parte del hipervínculo). Además, se cuidó tener una interfase limpia y que, al mismo tiempo que envía toda la información importante, resulte cómoda de navegar.

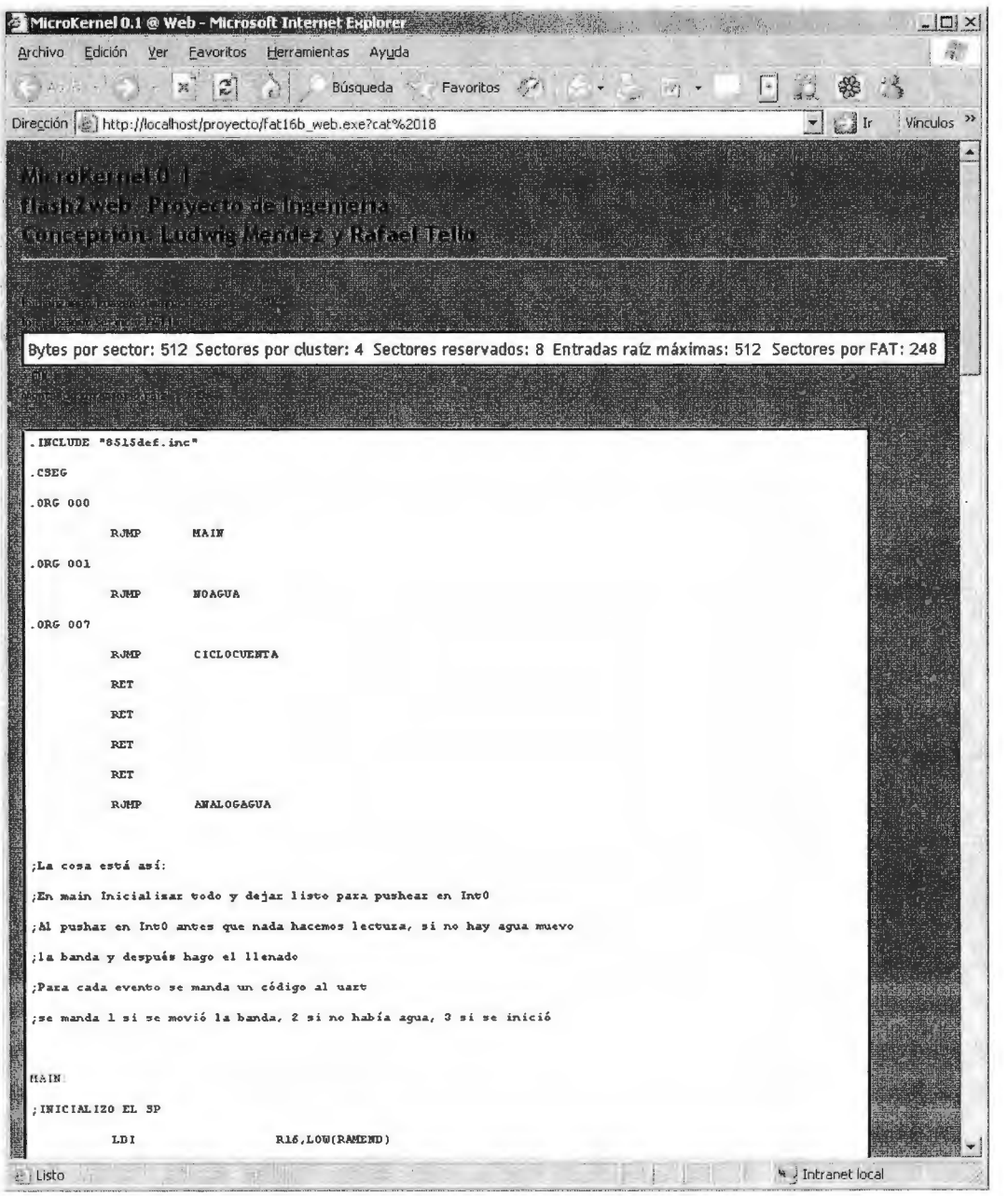


Imagen 4.1.2.2. Vista de un archivo de texto en el mismo programa

Actualmente se están desarrollando más funciones para este programa en específico, algunas de las cuales son una carpeta "administrativa", que permita subir archivos, formatear la unidad y tener visibilidad completa sobre la unidad (clusters libres, bytes libres, bytes dañados, etc). Estas funciones deberán estar listas para el día de la presentación final.

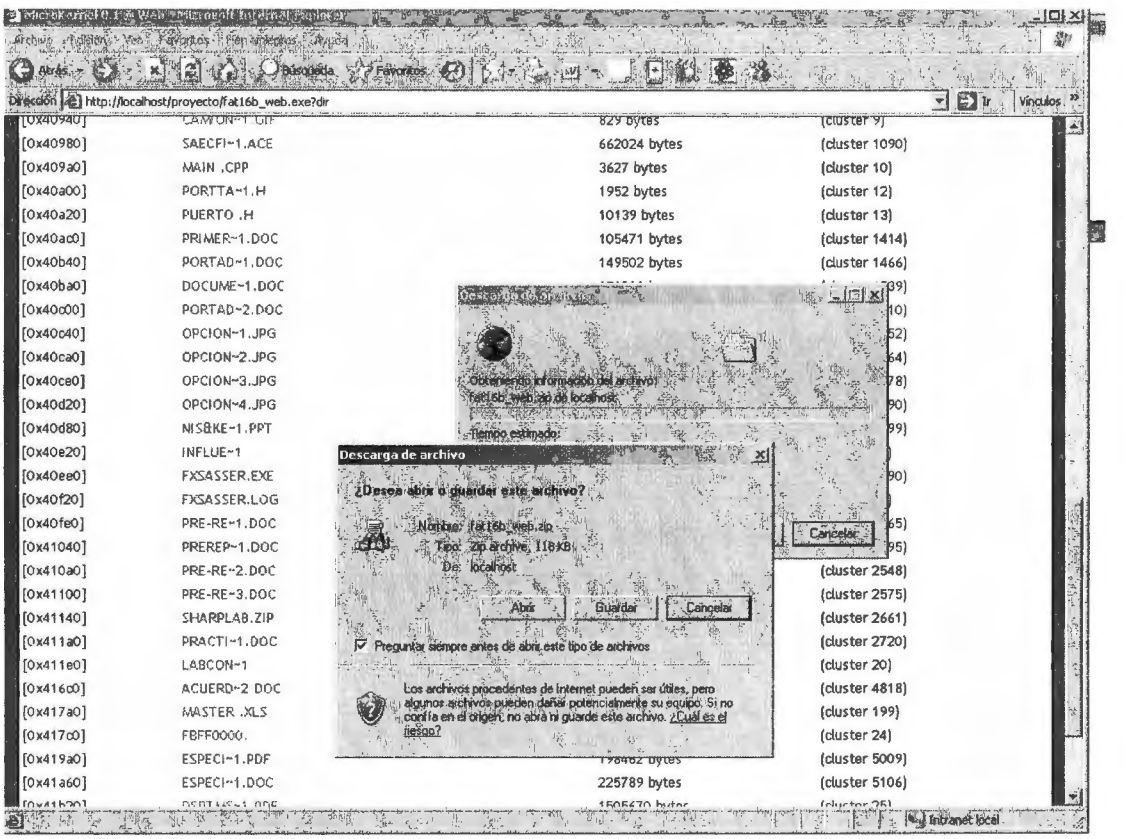


Imagen 4.1.2.3. Abriendo un archivo, y dejando a Internet Explorer la decisión de qué hacer.

4.1.3. Emulador de memoria serial

4.2. Proyecto integrado en Rabbit

4.2.1. Software

Los resultados obtenidos en el Rabbit fueron similares a los obtenidos en el servidor web para PC, con las siguientes limitaciones:

- Debido a la poca memoria del Rabbit, no fue posible implementar algún buffer que permitiera transmitir archivos binarios a través de la red al cliente, con el fin de descargar archivos. Por lo mismo, se desactivó la función de descargar todo tipo de archivos.

- Debido al limitado tamaño del buffer, se descubrió un error en la programación de la biblioteca http.lib, donde en caso de recibir una variable grande, más que el tamaño completo del buffer, se envía al cliente el volcado de toda la memoria RAM. Por lo mismo se ve severamente limitado el número máximo de archivos que se pueden mostrar en la pantalla. La discusión sobre esto podrá ser encontrada en Conclusiones.

4.2.2. Hardware

Sin duda este ha sido el punto más problemático del sistema, a pesar de que originalmente se había considerado que, al tratarse de protocolos de comunicación estándares, su implementación sería rápida. La necesidad de obtener resultados tangibles en este rubro, hizo necesario el desarrollo de alternativas diferentes a las inicialmente consideradas.

4.2.2.1. Comunicación con SD vía SPI

Por cuestión de facilidad de hardware, se decidió utilizar el protocolo SPI, en lugar del protocolo nativo de Secure Digital, ya que requiere de mucho más lógica si no se cuenta con el driver adecuado.

A partir de las palabras de control de la tarjeta, se desarrolló el algoritmo para hacer lecturas a la tarjeta en bloques de 1 byte y de 2048 bytes (tamaño del cluster para la mayoría de las memorias de entre 64 y 256 MB). Sin embargo no se logró la comunicación, debido a que el pin de RXB (MISO), se encuentra siempre en un estado alto, sin importar qué se le conecte. El mismo problema se presentó al intentar utilizar comunicación serial asíncrona con la computadora, a pesar de que el Rabbit cuenta con una interfaz para el RS232 que se conecta directamente al puerto serial de la computadora. Se levantó una solicitud de soporte técnico directamente con el fabricante, y se ha buscado por este problema similar en diferentes foros (incluyendo los moderados por staff de Rabbit Semiconductor).

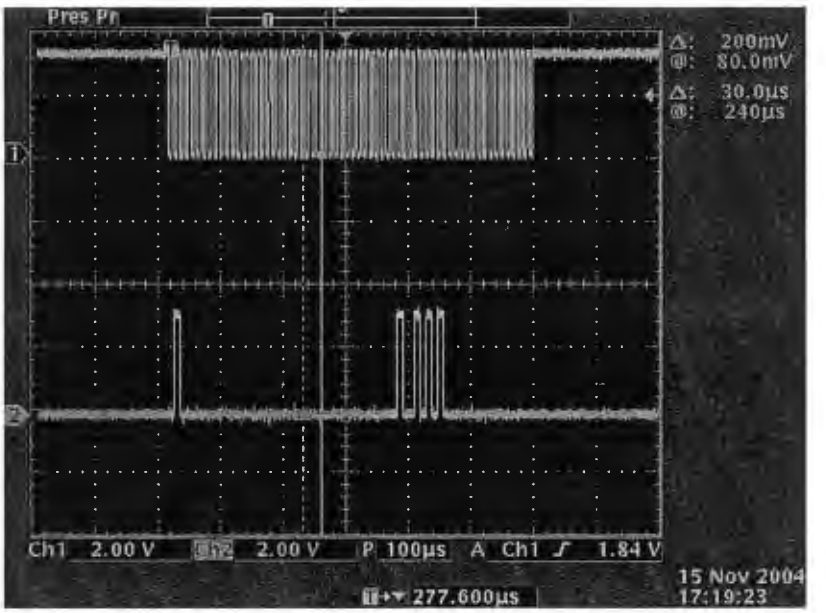


Imagen 7.2.1.1. Transmisión en TXB y CLKB de una palabra de control completa

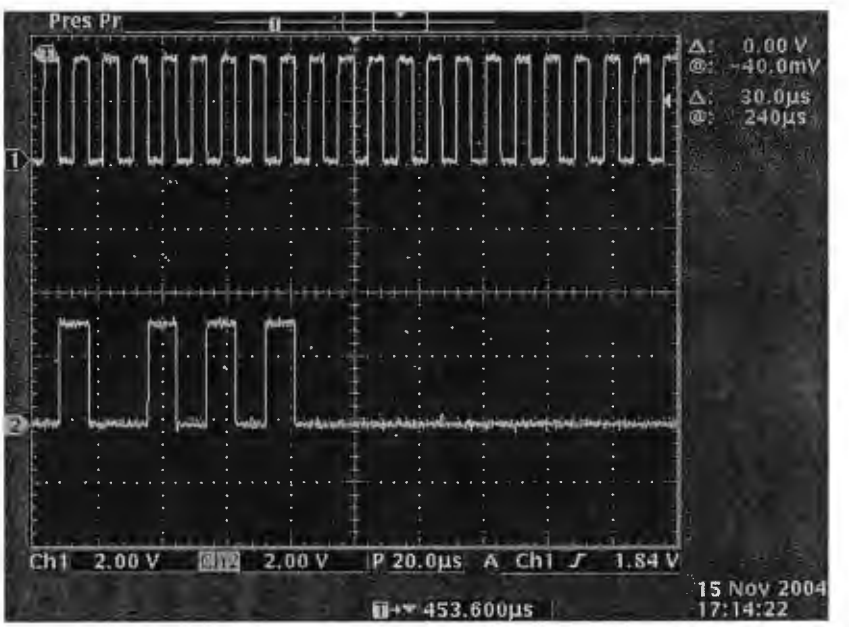


Imagen 4.2.2.2. Detalle de TXB y CLKB en el Rabbit

4.2.2.2. Comunicación de memoria Flash emulada vía serial + USB

Como se comentó anteriormente, se desarrolló un emulador de memoria Flash en la PC, utilizando 3 componentes principales. El primero, es el driver de bajo nivel provisto directamente por FTDI, y el cual está disponible sin costo en www.ftdichips.com o en www.dlpdesign.com (este driver no se documenta aquí, ya que escapa a los propósitos de este proyecto). El segundo es el DLL que sirve de enlace entre el API del driver/Win32, y el programa de alto nivel que se desee; el tercero es precisamente el programa de propósito específico que atenderá a las peticiones.

5. Conclusiones

A lo largo de este proyecto se descubrió que el cumplimiento de los objetivos del proyecto requeriría de mayor información y experimentación que la estimada inicialmente, debido a la poca información sobre la unificación de la lectura de FAT y el uso de bajo nivel de web en dispositivos embebidos. Sin embargo, al final del proyecto se lograron cumplir los objetivos en forma general, aunque con ciertas limitantes en la forma de resolver cada uno de los problemas de diseño. Debido a la naturaleza multienfoque del proyecto, se plantearán las conclusiones en función a cada uno de los objetivos.

Implementar un dispositivo con acceso a Ethernet que sea cliente DHCP y servidor http

Desarrollar una interfase para comunicar el dispositivo de acceso a Ethernet con el sistema de acceso al medio de almacenamiento

Durante el tiempo que se tuvo el Rabbit (restringido por los tiempos de entrega y retención en aduanas), se pudo constar que el Rabbit es un instrumento para realizar el control embebido en forma remota muy versátil, con muchas facilidades de programación, pero indudablemente con muchas oportunidades en cuanto a la mejora de bibliotecas se refiere. Desde las limitaciones con las estructuras dinámicas que obligaron a disminuir enormemente la capacidad inicial del programa, hasta los problemas derivados de reducir el tamaño del búfer TCP/IP afectaron gravemente el resultado final del servidor http del Rabbit, debido al limitado potencial de salida que de ello derivó. Por otro lado, se encontró que la documentación del Rabbit es buena, pero se encuentra bastante dispersa. Es posible encontrar notas de aplicación que indican pines no documentados en algún lugar del manual; es también posible encontrar en los foros del mismo fabricante, referencias a white papers que simplemente han sido removidos de la página. Sería bueno que proyectos subsecuentes que empleen el Rabbit, antes de comenzar el desarrollo se aseguren de documentarse lo más posible, y no confiar plenamente en el (los) manuales de usuario, que muchas veces incluso, se contradicen entre ellos mismos. No se tuvo el suficiente tiempo para realizar más revisiones, pero futuros trabajos con el Rabbit para este tipo de propósitos, deberían considerar el implementar manualmente, con menos funcionalidad pero mayor flexibilidad, el acceso directo a escribir en el servidor web, utilizando las estructuras básicas de http.lib (como el HttpState->buffer), y con ello tener control total sobre la información que sale, sin importar su tamaño.

Otra alternativa al problema, es el uso del Dynamic C 9.1; dentro de sus mejoras se encuentra el soporte a malloc, calloc, además de una compilación más óptima que la encontrada en el Dynamic C 8.1. En el caso de este equipo, se recibió un upgrade gratuito del Dynamic C, sin

embargo este fue entregado una semana antes de presentar el proyecto, por lo que se determinó inviable el modificar todo lo ya hecho hasta el momento.

La implementación del cliente DHCP no supuso mayor problema utilizando las funciones del Rabbit. Sobre lo estudiado no resulta del todo descabellado implementar un servidor DHCP que permita atender aquellas redes donde la conectividad/capacidad de agregado de nuevos equipos es limitada. Tal vez sería cuestión de estudiar más a fondo la biblioteca tcrtcp.lib, que es la que contiene estas funciones, para poder implementar el servicio DHCP Server al caducar la petición del cliente DHCP.

Generar el hardware y software necesarios para acceder al sistema de archivos de una memoria Flash, desde la interfase física con la tarjeta hasta el soporte para abstraer el sistema de archivos.

Indudablemente, el punto donde se lograron los mayores avances de investigación, y donde se pudo investigar con mayor profundidad los efectos de cada función es en el desarrollo de las funciones de acceso a FAT. Se implementó completamente la lectura de archivos de cualquier tamaño permitido por FAT16, la lectura de particiones primarias FAT16 de hasta 2GB sin ningún requerimiento adicional de código; se desarrollaron los algoritmos para realizar la escritura en FAT16, así como para realizar el formato lógico rápido y completo de toda unidad FAT16 y la defragmentación de disco. Para probar este sistema, se desarrollaron dos programas de emulación utilizando una imagen de disco generada binariamente, los cuales al realizar la lectura y escritura en forma independiente al algoritmo, y hacer indistinto para el resto del programa el leer de una memoria real a leer de una imagen virtual, se consideraron experimentos válidos. Proyectos subsecuentes podrían tomar como base el presente proyecto para implementar comandos de escritura en el sistema (como un COPY del sistema real al sistema virtual); el soporte para Long Filename Specification, que permitiría una interfaz con el usuario mucho más amigable; un GUI para realizar el formateo y la defragmentación de cualquier partición; el soporte para particiones extendidas FAT16 y para particiones FAT32. Se apreció durante el desarrollo que la información se encuentra sumamente dispersa debido al tiempo que ha transcurrido desde la implementación de FAT16, y al hecho que los sistemas operativos contemporáneos sólo utilizan NTFS o EXT3.

Las tarjetas Secure Digital son medios de almacenamiento altamente confiables, de mínimos requerimientos de energía y vida útil muy extensa. Las propiedades de compatibilidad para diferentes tipos de comunicación la hacen útil para diferentes sistemas sin necesidad de un diseño particular para la tarjeta SD, sino que el diseño solo tiene que cubrir el estándar internacional SPI y así puede utilizarse con la tarjeta SD o con muchos diferentes tipos de memoria más. Sin embargo, no obstante la disponibilidad de una hoja de especificaciones de la

tarjeta en Internet; la información contenida en la misma es escasa e inexacta, los productores son herméticos respecto a la información de sus productos con aquellos que no compran las especificaciones como diseñadores y, en general, los medios para obtener información sobre el diseño de comunicación con la tarjeta SD son deficientes. Un proyecto con más tiempo definitivamente podría utilizar un medio de almacenamiento mucho más robusto y con un estándar industrial probado, como podría ser la interfaz IDE o ATA de disco duro; o bien se podría realizar la conexión con más tipos de formato, como CompactFlash o Multimedia Card. Sin duda habría sido necesario realizar un análisis más profundo sobre el impacto de utilizar cada una de estas tecnologías, así como los riesgos que esto habría tenido en el cumplimiento de los objetivos del proyecto.

Por su parte, el estándar USB y más específicamente la interfaz entre USB y RS232, ha sido una tecnología empleada en el desarrollo de diversos proyectos anteriores. Esta tecnología ha venido a "salvar" en cierta medida los alcances del proyecto, al permitir demostrar la capacidad de comunicación del equipo Rabbit, más allá de los problemas que se han tenido con SPI. Sin duda deberá ser una solución provisional, en lo que se encuentra una solución definitiva a las limitaciones en comunicación. Ya sea que se utilice Secure Digital, MMC, CF, Memory Stick o incluso una pluma USB, sería muy importante para el proyecto el lograr la comunicación con alguno de ellos.

6. Referencias bibliográficas

1. Axelson, Jan. USB Complete. Lakeview Research LLC, 2002
2. Axelson, Jan. Embedded Ethernet and Internet Complete. Lakeview Research LLC, 2003.
3. Cyliax, Ingo. "FTDI USB Adapter Interfacing for Rabbit" http://www.zworld.com/documentation/docs/refs/AN405/AN405.htm, rev. ago. 2003
4. RCM3200 RabbitCore User's Manual, www.rabbitsemiconductor.com
5. "SD Card Specification". http://www.sandisk.com/pdf/oem/SD_Physical_specsv101.pdf . SD Group, 2002
6. Dobiash, Jack. "FAT16 Structure Information". http://home.teleport.com/~brainy/fat16.htm, Rev. 17/06/99.
7. "FAT32". Win32 Development: Windows Platform SDK: Storage. Microsoft Development Network. Rev. Nov. 2004.
8. "FAT16 and FAT12". Win32 Development: Windows Platform SDK: Storage. Microsoft Development Network. Rev. Nov. 2004.
9. Dobiash, Jack. "FAT32 Structure Information". http://home.teleport.com/~brainy/fat16.htm, Rev. 17/06/99
10. "SD card, the concept". SD Card Forum. http://www.sdcard.com/usa/TextPage.asp?Page=1
11. "About the SD Memory Card", 2004 SD Card Association. http://www.sdcard.org/sd_memorycard/index.html
12. Shiftlett, Chris. "HTTP developer's handbook". Indianapolis Imp. 2003
13. Marshall, James. "HTTP made really easy". http://www.jmarshall.com/easy/http/. Rev.03/03/04
14. Smith, George. "Computer Interfacing". Oxford Press, 2000
15. Kalinsky, David, Kalinsky Roee. "Introduction to Serial Peripheral Interface". http://www.embedded.com/story/OEG20020124S0116, Rev. 01/02/02
16. Wilson, G.R. Embedded Systems and Computer Architecture. Boston, NewNess, 1a ed. 2000
17. Berger, Arnold. Embedded systems design: an introduction to processes, tools and techniques. Berkeley CA. Group West. 2002
18. Future Technology Devices Intl. Ltd. FTD2XX Programmer's Guide Version 2.01, http://www.dlpdesign.com/d2xxpg21.pdf, rev. Nov 2002.
19. vinDaci. "Long Filename Specification". http://home.teleport.com/~brainy/lfn.html, 6 Ene. 1998

7. Anexos

7.1. Emulador FAT en PC. Código fuente.

```
#include <stdio.h>

typedef struct archv{
      char nombre[12];
   short esDir;
   unsigned long tamanio;
   char fcreacion[11];
   unsigned int dirInicio;
   unsigned int dirEntry;
   short abierto;
} archivo;

int clustersArchivo[256];

FILE *miArch;

archivo archivos[128];
archivo curDirArch;
char readTemp[2049];

unsigned int
bytesXSector,sectoresXCluster,entradasRaiz,sectoresReservados,sectoresXFAT,num
eroFATs;
unsigned int offsetParticion,offsetFAT,offsetRaiz,offsetData,dirActual;
unsigned short masArchivos;

unsigned char leeByte(unsigned int direccion)
{
    char temp;
    char *t = &temp;
      fseek(miArch,direccion,SEEK_SET);
      fread((void*)t,1,1,miArch);
      return temp;
}
void escribeByte(unsigned int direccion, unsigned char dato)
{
    char *t = &dato;
    fseek(miArch,direccion,SEEK_SET);
    fwrite((void*)t,1,1,miArch);
}
unsigned int leeInt(unsigned int direccion)
{
      return (unsigned int)(leeByte(direccion))+(unsigned
int)(256*leeByte(direccion+1));
}

unsigned char* leeString(unsigned int direccion, int tamanio){
    int i;
    if(tamanio>2048) tamanio = 2048;
    for(i=0;i<tamanio;i++){
        readTemp[i]=leeByte(direccion+i);
    }
    readTemp[i] = '\0';
    return readTemp;
}

archivo* myfopen(archivo *arch, int offsetCluster)
{
    int i;
```

```
    printf("\nInicializando apertura de archivo %s",arch->nombre);
    if(offsetCluster==0) clustersArchivo[0] = arch->dirInicio;
    else clustersArchivo[0] = offsetCluster;
    printf("\n Clusters del archivo: %d,",clustersArchivo[0]);
    for(i=1;i<256;i++)
    {
        clustersArchivo[i] = leeInt(offsetFAT+2*clustersArchivo[i-1]);
        printf("%d",clustersArchivo[i]);
        if(clustersArchivo[i]>=0xFFF8) break;
        printf(", ");
    }
    printf("\n");
    arch->abierto = 1;
    return arch;
}

void myfclose(archivo *arch)
{
    int i;
    if(arch->abierto==1)
    {
        for(i=0;i<256;i++)
        {
            clustersArchivo[i] = 0;
        }
        arch->abierto = 0;
    }
}


void dir()
{
    archivo *arch;
    int x;
    arch = &curDirArch;

    printf("Leyendo directorio %s\n",arch->nombre);
        //Despliegue
    for(x=0;x<128;x++)
    {
        if(archivos[x].nombre[0]=='\0') break;
        printf("[0x%x] ",archivos[x].dirEntry);
        printf("%d) %s %s %d bytes   (cluster
%d)\n",x,archivos[x].nombre,((archivos[x].esDir==1)?" <DIR> ":"
"),archivos[x].tamanio,archivos[x].dirInicio);
    }


}
unsigned int format(char rapidoOcompleto)
{
    //Instrucción que realiza el formateo lógico en la unidad
    //Paso 1: limpia todos los clusters con estado diferente a 0xFFF7
    //Paso 2: Elimina todas las entradas del rootDirectory
    //Paso 3: Escribe 0x00h en todas las localidades de datos si se eligió
rápido
    printf("Iniciando formato de unidad... ");

}
unsigned int mycopy(archivo *arch,char *destino)
{
    int x,i,ap,ncl;
    long bytesLeidos;
    FILE *archW32;
    x=0;
    if(arch->abierto==1)
    {
```

```
        bytesLeidos = 0;
        archW32 = fopen(destino,"wb");
        ncl = arch->tamanio / bytesXSector*sectoresXCluster;
        if(archW32==NULL) return 1;
        do
        {
            ap = offsetData+(clustersArchivo[x]-
2)*bytesXSector*sectoresXCluster;
            if(ncl==x)
            {
                fwrite(leeString(ap,arch->tamanio-bytesLeidos),arch->tamanio-
bytesLeidos,1,archW32);
                break;
            }
            else
fwrite(leeString(ap,bytesXSector*sectoresXCluster),bytesXSector*sectoresXClust
er,1,archW32);
          x++;
          if(x==255)
          {
              myfopen(arch,clustersArchivo[x]);
              x = 0;
          }
        } while(clustersArchivo[x]<0xFFF8);

    }
}
unsigned int mycat(archivo *arch)
{
    int x,i,ap,ncl;
    long bytesLeidos;
    x=0;
    if(arch->abierto==1)
    {
        bytesLeidos = 0;
        ncl = arch->tamanio / bytesXSector*sectoresXCluster;
        do
        {
            ap = offsetData+(clustersArchivo[x]-
2)*bytesXSector*sectoresXCluster;
            //printf("Para cluster %d inicializo apuntador en
%d",clustersArchivo[x],ap);
            if(ncl==x)
            {
                printf("%s",leeString(ap,arch->tamanio-bytesLeidos));
                break;
            }
            else
            {
                printf("%s",leeString(ap,bytesXSector*sectoresXCluster));
            }
          x++;
          if(x==255)
          {
              myfopen(arch,clustersArchivo[x]);
              x = 0;
          }
        } while(clustersArchivo[x]<0xFFF8);

    }
}
void cd(archivo *arch)
{
    unsigned long ap,x;
    char fecha[16];
    char tmp[4];
      if(arch->esDir)
```

```
    {
        //Sí es directorio, hacer el cambio
        if(arch->dirInicio==0){
            ap = offsetRaiz;
            dirActual = offsetRaiz;
            curDirArch = *arch;
        }
        else
        {
            ap = offsetData + (arch->dirInicio-
2)*bytesXSector*sectoresXCluster;
            dirActual = ap;
            curDirArch = *arch;
        }

        for(x=0;x<128;x++)
        {
            if((leeByte(ap)==0x00)||((dirActual!=offsetRaiz)&&(ap-
dirActual>=bytesXSector*sectoresXCluster))){
                strcpy(archivos[x].nombre,"\0");
                break;
            }
            if((leeByte(ap)==0xE5)||(leeByte(ap+11)==0x0F)){
                printf("[@%lx] Ignorando 1 elemento eliminado... \n",ap);
                ap+=32;
                x--;
                continue;
            }
            if(((leeByte(ap+11))&0x10)>0)
            {

                archivos[x].esDir = 1;
            }
            else
            {
                archivos[x].esDir = 0;
            }

            //Entrada directorio
            archivos[x].dirEntry = ap;
            //Nombre
            strcpy(archivos[x].nombre,leeString(ap,8));
            if(archivos[x].esDir==0) strcat(archivos[x].nombre,".");
            if(archivos[x].esDir==0)
strcat(archivos[x].nombre,leeString(ap+8,3));
            //printf("[@%lx] Item %d: %s\n",ap,x,archivos[x].nombre);
            //¿Es directorio?

            //Cluster de inicio
            archivos[x].dirInicio = leeInt(ap+26);

            //Fecha (y hora...algún día)

            ltoa((leeInt(ap+0x18)&0x001F),tmp); //Día
            strcpy(fecha,tmp);
            ltoa((leeInt(ap+0x18)&0x01E0)>>5,tmp); //Mes
            strcat(fecha,"/");
            strcat(fecha,tmp);
            ltoa(((unsigned int)(leeInt(ap+0x18)&0xFE00))>>9+1980,tmp); //Año
            strcat(fecha,"/");
            strcat(fecha,tmp);
            strcpy(archivos[x].fcreacion,fecha);

            //Tamaño
            archivos[x].tamanio = leeInt(ap+28)+leeInt(ap+30)*0xFFFF;

            //Ver si hay más
```

```
            if(leeByte(ap+32)==0)
            {
                strcpy(archivos[x+1].nombre,"\0");
                break;
            }
            else
            {
                //Caso normal
                ap+=32;
            }
        }
    }
}

void inicializa()
{
    if(leeByte(16)){
        bytesXSector = leeInt(11); //11 y 12
        sectoresXCluster = (unsigned int)leeByte(13); //sólo 13
        sectoresReservados = leeInt(14); //14 y 15
        //16 es number of FAT's
        entradasRaiz = leeInt(17); //17 y 18
        //19 y 20 son para #sectores, 21 para media descr.
        sectoresXFAT = leeInt(22);
        offsetParticion = 0;
        offsetFAT = offsetParticion + sectoresReservados*bytesXSector;
        offsetRaiz = offsetFAT+2*sectoresXFAT*bytesXSector;
        offsetData = offsetRaiz+entradasRaiz*32;
        dirActual = offsetRaiz;
        curDirArch.esDir = 1;
        strcpy(curDirArch.nombre,"/");
        curDirArch.dirInicio = 0;
        printf("Bytes por sector: %d\n",bytesXSector);
        printf("Sectores por cluster: %d\n",sectoresXCluster);
        printf("Sectores reservados: %d\n",sectoresReservados);
        printf("Entradas raíz máximas: %d\n",entradasRaiz);
        printf("Sectores por FAT: %d\n",sectoresXFAT);
        printf("Offset Directorio Raíz: %lx\n",offsetRaiz);
        printf("Offset FAT1: %lx\n",offsetFAT);
    }
    else{
        exit(1);
    }
}

int main(char argv[], int argc)
{
    char prompt[64];
    char comando[64];
    archivo *tmp;
    int num;
    strcpy(prompt,"USBDrive#>");
    printf("MicroKernel 0.1. \n\n");
    printf("flash2web. Proyecto de Ingeniería\n");
    printf("Concepción: Ludwig Mendez y Rafael Tello\n\n- - - - - - - - - -
- - \n");
    printf("Inicializando imagen binaria de disco...\n");
    if((miArch = fopen("miusb.whx","rb"))==NULL)
    {
        printf("<ERROR>\n");
        exit(1);
    }
    printf("\n                                    <  OK  >\n");
    printf("Inicializando servicio FAT16...\n");
    inicializa();
    printf("\n                                    <  OK  >\n");
```

```
    num = 0;
    printf("Montando directorio raiz...      ");
    cd(&curDirArch);
    printf("<  OK  >\n\n");
    printf("Inicializando prompt...          <  OK  >\n");
    while(1){
        printf("\n%s",prompt);
        scanf("%s",comando);

        if((strcmpi(comando,"dir")==0)||(strcmpi(comando,"ls")==0))
        {
            //Usuario tecleó dir... mostramos directorio raíz
            dir();
        }
        else if((strcmpi(comando,"cd")==0))
        {
            printf("\n\nIntroduzca el número de directorio: ");
            scanf("%d",&num);
            printf("\nCambiando a directorio %s en cluster
%d",archivos[num].nombre,archivos[num].dirInicio);
            if(archivos[num].esDir==1)
            {
                //printf("%s",archivos[num].nombre);
                prompt[strlen(prompt)-2] = '\\';
                prompt[strlen(prompt)-1] = '\0';
                strcat(prompt,archivos[num].nombre);
                curDirArch = archivos[num];
                strcat(prompt,"#>");
                cd(&archivos[num]);

            }
            else
            {
                printf("\nError: %s no es un
directorio.\n",archivos[num].nombre);
            }
        }
        else if((strcmpi(comando,"copy")==0)||(strcmpi(comando,"cp")==0))
        {
            printf("\n\nIntroduzca el número de archivo: ");
            scanf("%d",&num);
            printf("\nTeclee nombre de archivo destino (formato LFN
aceptado)");
            scanf("%s",comando);
            if(archivos[num].esDir==0)
            {
                tmp = myfopen(&archivos[num],0);
                mycopy(tmp,comando);
            }
            else
            {
                printf("\nError: %s es un
directorio.\n",archivos[num].nombre);
            }
        }
        else if((strcmpi(comando,"cat")==0)||(strcmpi(comando,"type")==0))
        {
            printf("\n\nIntroduzca el número de archivo: ");
            scanf("%d",&num);
            if(archivos[num].esDir==0)
            {
                tmp = myfopen(&archivos[num],0);
                mycat(tmp);
                return 0;
            }
            else
            {
```

```
                printf("\nError: %s es un
directorio.\n",archivos[num].nombre);
             }
         }
         else printf("Comando o nombre de archivo no valido");
    }
       return 0;
}
```

7.2. Emulador FAT en Web (PC). Código fuente.

```
#include <stdio.h>

typedef struct archv{
      char nombre[12];
   short esDir;
   unsigned long tamanio;
   char fcreacion[11];
   unsigned int dirInicio;
   unsigned int dirEntry;
   short abierto;
} archivo;

int clustersArchivo[256];

FILE *miArch;

archivo archivos[128];
archivo curDirArch;
char readTemp[2049];

unsigned int
bytesXSector,sectoresXCluster,entradasRaiz,sectoresReservados,sectoresXFAT,num
eroFATs;
unsigned int offsetParticion,offsetFAT,offsetRaiz,offsetData,dirActual;
unsigned short masArchivos;
//Declaración de funciones

void miHttp_init();
unsigned char leeByte(unsigned int);
unsigned int leeInt(unsigned int);
unsigned char* leeString(unsigned int,unsigned int);
void cd(archivo*);
void inicializa();

void miHttp_init()
{
    printf("HTTP/1.1 200 OK\n");
    printf("Content-type: text/html\n");
    printf("Server: MicroKernel 0.1\n\n");
    printf("<html><head><title>MicroKernel 0.1 @ Web</title>\n");
    printf("    <style type='text/css'>\n");
    printf("<!-- \n");
    printf("body {\n");
    printf("        background-color: #006699;\n");
    printf("        font-size: x-small;\n");
    printf("        font-family: 'Trebuchet MS', Arial;\n");
    printf("}\n");
    printf("a:link {\n");
    printf("color: #FF0000;\n");
    printf("text-decoration: none;\n");
    printf("}\n");
    printf("a:visited {\n");
    printf("color: #FF0000;\n");
    printf("text-decoration: none;\n");
    printf("}\n");
```

```
    printf("a:hover {\n");
    printf("color: #FF0000;\n");
    printf("text-decoration: none;\n");
    printf("}\n");
    printf("p {\n");
    printf("        font-size: xx-small;\n");
    printf("}\n");
    printf("table {\n");
       printf("background-color: FFFFFF;\n");
       printf("border: thin solid #000000;\n");
    printf("        font-size: x-small;\n");
       printf("}\n");
    printf("-->\n</style></head><body>\n");
    printf("<h3>MicroKernel 0.1. <br>\n");
    printf("flash2web. Proyecto de Ingeniería<br>");
    printf("Concepción: Ludwig Mendez y Rafael Tello<hr></h3>\n");
    printf("<p>Inicializando imagen binaria de disco...\n");
    if(!ImageOpen()){
        printf("<ERROR><br>");
        exit(1);
    }
    printf("\n                                    <  OK  ><br>\n");
    printf("Inicializando servicio FAT16... ");
    inicializa();
    printf("<table width='100%%'><tr>\n");
       printf("<td>Bytes por sector: %d</td>\n ",bytesXSector);
       printf("<td>Sectores por cluster: %d</td>\n ",sectoresXCluster);
       printf("<td>Sectores reservados: %d</td>\n ",sectoresReservados);
       printf("<td>Entradas raíz máximas: %d</td>\n ",entradasRaiz);
       printf("<td>Sectores por FAT: %d</td></tr></table>\n ",sectoresXFAT);
    printf(" <  OK  ><br>");
    printf("Montando directorio raiz...     ");
    printf("<  OK  ></p>");
}

unsigned char leeByte(unsigned int direccion)
{
    char temp;
    char *t = &temp;
       fseek(miArch,direccion,SEEK_SET);
       fread((void*)t,1,1,miArch);
       return temp;
}
void escribeByte(unsigned int direccion, unsigned char dato)
{
    char *t = &dato;
    fseek(miArch,direccion,SEEK_SET);
    fwrite((void*)t,1,1,miArch);
}
unsigned int leeInt(unsigned int direccion)
{
//    printf("(Leo %d en %d y %d en %d)",
       return (unsigned int)(leeByte(direccion))+(unsigned
int)(256*leeByte(direccion+1));
}

unsigned char* leeString(unsigned int direccion, unsigned int tamanio){
    int i;
    if(tamanio>2048) tamanio = 2048;
    for(i=0;i<tamanio;i++){
        readTemp[i]=leeByte(direccion+i);
    }
    readTemp[i] = '\0';
    return readTemp;
}

archivo* myfopen(archivo *arch, int offsetCluster)
```

```
{
    int i;
    if(offsetCluster==0) clustersArchivo[0] = arch->dirInicio;
    else clustersArchivo[0] = offsetCluster;
    for(i=1;i<256;i++)
    {
        clustersArchivo[i] = leeInt(offsetFAT+2*clustersArchivo[i-1]);
        if(clustersArchivo[i]>=0xFFF8) break;
    }
    arch->abierto = 1;
    return arch;
}

void myfclose(archivo *arch)
{
    int i;
    if(arch->abierto==1)
    {
        for(i=0;i<256;i++)
        {
            clustersArchivo[i] = 0;
        }
        arch->abierto = 0;
    }
}

void dir()
{
    archivo *arch;
    int x;
    arch = &curDirArch;

    printf("Leyendo directorio %s\n<br>",arch->nombre);
    printf("<table width='100%%'><tr><td>Dir. Entry</td><td>Nombre de
archivo</td><td>Tamaño</td><td>Cluster inicio</td></tr>");
    if(arch->nombre[0]!='/') printf("<tr><td> </td><td align='center'><a
href='javascript:history.go(-1);'><b>Volver a directorio
anterior</b></a></td><td> </td><td> </td></tr>\n");
        //Despliegue
    for(x=0;x<128;x++)
    {
        if(archivos[x].nombre[0]=='\0') break;
        if(archivos[x].nombre[0]=='.') continue;
        printf("<tr>");
        printf("<td>[0x%x] </td>",archivos[x].dirEntry);
        if(archivos[x].esDir==1)
        {
            printf("<td><a href='fat16b_web.exe?cd
%d'>%s</a></td><td>%s</td><td>%d bytes  </td><td> (cluster
%d)</td></tr>\n",x,archivos[x].nombre,((archivos[x].esDir==1)?" <DIR> ":"
"),archivos[x].tamanio,archivos[x].dirInicio);
        }
        else{
            printf("<td><a target='_blank' href='fat16b_web.exe?cat %d'> %s
</td><td> %s </td><td>%d bytes  </td><td> (cluster
%d)</td></tr>\n",x,archivos[x].nombre,((archivos[x].esDir==1)?" <DIR> ":"
"),archivos[x].tamanio,archivos[x].dirInicio);
        }
    }
    if(arch->nombre[0]!='/') printf("<tr><td> </td><td align='center'><a
href='javascript:history.go(-1);'><b>Volver a directorio
anterior</b></a></td><td> </td><td> </td></tr>\n");
    printf("</table>");
}
unsigned int format(char rapidoOcompleto)
{
    //Instrucción que realiza el formateo lógico en la unidad
```

```
    //Paso 1: limpia todos los clusters con estado diferente a 0xFFF7
    //Paso 2: Elimina todas las entradas del rootDirectory
    //Paso 3: Escribe 0x00h en todas las localidades de datos si se eligió
rápido
    printf("Iniciando formato de unidad... ");

}
unsigned int mycopy(archivo *arch,char *destino)
{
    int x,i,ap,ncl;
    long bytesLeidos;
    FILE *archW32;
    x=0;
    if(arch->abierto==1)
    {
        bytesLeidos = 0;
        archW32 = fopen(destino,"wb");
        ncl = arch->tamanio / bytesXSector*sectoresXCluster;
        if(archW32==NULL) return 1;
        do
        {
            ap = offsetData+(clustersArchivo[x]-
2)*bytesXSector*sectoresXCluster;
            if(ncl==x)
            {
                fwrite(leeString(ap,arch->tamanio-bytesLeidos),arch->tamanio-
bytesLeidos,1,archW32);
                break;
            }
            else
fwrite(leeString(ap,bytesXSector*sectoresXCluster),bytesXSector*sectoresXClust
er,1,archW32);
          x++;
          if(x==255)
          {
              myfopen(arch,clustersArchivo[x]);
              x = 0;
          }
        } while(clustersArchivo[x]<0xFFF8);

    }
}

unsigned char obtenMIME(archivo *arch)
{
    /*if(strcmpi(arch->nombre+9,"gif")==0) return 11;
    if((strcmpi(arch->nombre+9,"jpg")==0)||(strcmpi(arch->nombre+8,"jpe")==0))
return 10;
    if(strcmpi(arch->nombre+9,"png")==0) return 12;*/
    if(strcmpi(arch->nombre+9,"htm")==0) return 3;
    if(strcmpi(arch->nombre+9,"asm")==0) return 2;
    if(strcmpi(arch->nombre+9,"txt")==0) return 2;
    if(strcmpi(arch->nombre+9,"c  ")==0) return 2;
    if(strcmpi(arch->nombre+9,"cpp")==0) return 2;
    if(strcmpi(arch->nombre+9,"jav")==0) return 2;
    if(strcmpi(arch->nombre+9,"cs ")==0) return 2;
    return 0;
}

unsigned int mycat(archivo *arch)
{
    switch(obtenMIME(arch)){
        case 0:
            {
                char tmp[20];
                strcpy(tmp,"tmp/");
                strcat(tmp,arch->nombre);
```

```
                mycopy(arch,tmp);
                printf("HTTP/1.1 302 Object Moved\n");
                printf("Location: /proyecto/tmp/%s\n",arch->nombre);
                printf("Content-type: text/html\n\n");
                printf("<head><title>Object
moved</title></head>\n<body><h1>Object Moved</h1>This object may be found <a
HREF="">here</a>.</body>");
                return 0;
            }
            break;
        case 10:
            printf("HTTP/1.1 200 OK\n");
            printf("Content-type: image/jpeg\n");
            break;
        case 11:
            printf("HTTP/1.1 200 OK\n");
            printf("Content-type: image/gif\n");
            break;
        case 12:
            printf("HTTP/1.1 200 OK\n");
            printf("Content-type: image/png\n");
            break;
        case 2:
            miHttp_init();
            printf("<table><tr><td><pre size='3'>");
            break;
        case 3:
            printf("HTTP/1.1 200 OK\n");
            printf("Content-type: text/html\n");
            break;
    }
    int x,i,ap,ncl;
    long bytesLeidos;
    x=0;
    if(arch->abierto==1)
    {
        bytesLeidos = 0;
        ncl = arch->tamanio / bytesXSector*sectoresXCluster;
        do
        {
            ap = offsetData+(clustersArchivo[x]-
2)*bytesXSector*sectoresXCluster;
            if(ncl==x)
            {
                printf("%s",leeString(ap,arch->tamanio-bytesLeidos));
                break;
            }
            else
            {
                printf("%s",leeString(ap,bytesXSector*sectoresXCluster));
            }
          x++;
          if(x==255)
          {
              myfopen(arch,clustersArchivo[x]);
              x = 0;
          }
        } while(clustersArchivo[x]<0xFFF8);
        if(obtenMIME(arch)==2) printf("</pre></td></tr></table>");
        else printf("%c",0x1A);
    }
}

void cd(archivo *arch)
{
    int ap,x;
    char fecha[16];
```

```
    char tmp[4];
       if(arch->esDir)
    {
        //Sí es directorio, hacer el cambio
        if(arch->dirInicio==0){
            ap = offsetRaiz;
            dirActual = offsetRaiz;
            curDirArch = *arch;
        }
        else
        {
            ap = offsetData + (arch->dirInicio-
2)*bytesXSector*sectoresXCluster;
            dirActual = ap;
            curDirArch = *arch;
        }

        for(x=0;x<128;x++)
        {
            if((leeByte(ap)==0x00)||((dirActual!=offsetRaiz)&&(ap-
dirActual>=bytesXSector*sectoresXCluster))){
                strcpy(archivos[x].nombre,"\0");
                break;
            }
            if((leeByte(ap)==0xE5)||(leeByte(ap+11)==0x0F)){
                ap+=32;
                x--;
                continue;
            }
            if(((leeByte(ap+11))&0x10)>0)
            {

                archivos[x].esDir = 1;
            }
            else
            {
                archivos[x].esDir = 0;
            }

            //Entrada directorio
            archivos[x].dirEntry = ap;
            //Nombre
            strcpy(archivos[x].nombre,leeString(ap,8));
            if(archivos[x].esDir==0) strcat(archivos[x].nombre,".");
            if(archivos[x].esDir==0)
strcat(archivos[x].nombre,leeString(ap+8,3));
            //¿Es directorio?

            //Cluster de inicio
            archivos[x].dirInicio = leeInt(ap+26);

            //Fecha (y hora...algún día)

            ltoa((leeInt(ap+0x18)&0x001F),tmp); //Día
            strcpy(fecha,tmp);
            ltoa((leeInt(ap+0x18)&0x01E0)>>5,tmp); //Mes
            strcat(fecha,"/");
            strcat(fecha,tmp);
            ltoa(((unsigned int)(leeInt(ap+0x18)&0xFE00))>>9+1980,tmp); //Año
            strcat(fecha,"/");
            strcat(fecha,tmp);
            strcpy(archivos[x].fcreacion,fecha);

            //Tamaño
            archivos[x].tamanio = leeInt(ap+28)+leeInt(ap+30)*0xFFFF;

            //Ver si hay más
```

```
            if(leeByte(ap+32)==0)
            {
                strcpy(archivos[x+1].nombre,"\0");
                break;
            }
            else
            {
                //Caso normal
                ap+=32;
            }
        }
    }
}

void inicializa()
{
    if(leeByte(16)){
        bytesXSector = leeInt(11); //11 y 12
        sectoresXCluster = (unsigned int)leeByte(13); //sólo 13
        sectoresReservados = leeInt(14); //14 y 15
        //16 es number of FAT's
        entradasRaiz = leeInt(17); //17 y 18
        //19 y 20 son para #sectores, 21 para media descr.
        sectoresXFAT = leeInt(22);
        offsetParticion = 0;
        offsetFAT = offsetParticion + sectoresReservados*bytesXSector;
        offsetRaiz = offsetFAT+2*sectoresXFAT*bytesXSector;
        offsetData = offsetRaiz+entradasRaiz*32;
        dirActual = offsetRaiz;
        curDirArch.esDir = 1;
        strcpy(curDirArch.nombre,"/");
        curDirArch.dirInicio = 0;
        cd(&curDirArch);
    }
    else{
        exit(1);
    }
}


int ImageOpen(){
    if ((miArch = fopen("miusb.whx","rb"))==NULL)
    {
        return 0;
    }
    else
        return 1;
}
int main(int argc, char *argv[])
{
    char prompt[64];
    char comando[64];
    archivo *tmp;
    int num;
    num = 0;
    if(argc > 1){ //Se solicitó un comando, el que sea
        strcpy(comando,argv[1]);
        if((strcmpi(comando,"dir")==0)||(strcmpi(comando,"ls")==0))
        {
            miHttp_init();
            dir();
        }
        else if((strcmpi(comando,"cd")==0))
        {
            miHttp_init();
            num = atoi(argv[2]);
```

```
            printf("<br>\nCambiando a directorio %s en cluster
%d<br>",archivos[num].nombre,archivos[num].dirInicio);
            if(archivos[num].esDir==1)
            {
                //printf("%s",archivos[num].nombre);
                prompt[strlen(prompt)-2] = '\\';
                prompt[strlen(prompt)-1] = '\0';
                strcat(prompt,archivos[num].nombre);
                curDirArch = archivos[num];
                strcat(prompt,"#>");
                cd(&archivos[num]);
                dir();
            }
            else
            {
                printf("\nError: %s no es un
directorio.\n",archivos[num].nombre);
            }
        }
        else if((strcmpi(comando,"copy")==0)||(strcmpi(comando,"cp")==0))
        {
            printf("\n\nIntroduzca el número de archivo: ");
            scanf("%d",&num);
            printf("\nTeclee nombre de archivo destino (formato LFN
aceptado)");
            scanf("%s",comando);
            if(archivos[num].esDir==0)
            {
                tmp = myfopen(&archivos[num],0);
                mycopy(tmp,comando);
            }
            else
            {
                printf("\nError: %s es un
directorio.\n",archivos[num].nombre);
            }
        }
        else if((strcmpi(comando,"cat")==0)||(strcmpi(comando,"type")==0))
        {
            ImageOpen();
            inicializa();
            num = atoi(argv[2]);
            if(archivos[num].esDir==0)
            {
                tmp = myfopen(&archivos[num],0);
                mycat(&archivos[num]);
                return 0;
            }
            else
            {
                miHttp_init();
                printf("\nError: %s es un
directorio.\n",archivos[num].nombre);
            }
        }
        else printf("%s: Comando o nombre de archivo no valido",argv[1]);
    }
    else{ //Se mandó llamar así nomás
       miHttp_init();
       printf("<h1>Próximamente un Flash muy mamón</h1>");
       printf("<a href='fat16b_web.exe?dir'>Navegar directorio raíz</a>");
    }
    fclose(miArch);
      return 0;
}
```

7.3. Programa final en Rabbit. Código fuente.

```
#class auto
#define TCPCONFIG 5
#define HTTP_MAXBUFFER 1024
#define HTTP_MAXSERVERS 1
#define MAX_TCP_SOCKET_BUFFERS 1
#define TCP_MAXPENDING 5
#memmap xmem
#use "dcrtcp.lib"
#use "http.lib"

#ximport "index.shtml"    index_html
const long maxBuffAdd = 256;
/*
-------------------------------------------------------------------------------
-
I N I C I A L I Z A C I O N E S
-------------------------------------------------------------------------------
-

*/
char dirText[1024];
const HttpType http_types[] =
{
       { ".shtml", "text/html", shtml_handler},
   { ".css", "text/css", NULL},


};

char curDir[256];
char curIP[16];
char redirectTo[256];
long currIP;
char stateURL[256];
char tmp[256];

const unsigned char initCmd[] = {0x40,0x00,0x00,0x00,0x00,0x95};

typedef struct archv{
       char nombre[13];
   short esDir;
   unsigned long tamanio;
   char fcreacion[11];
   unsigned long dirInicio;
   unsigned long dirEntry;
   short abierto;
} archivo;

int clustersArchivo[256];

archivo archivos[128];
archivo curDirArch;

//Nuevo: leer cluster entero
unsigned char bloqueAbs[512];
unsigned int bloqueAbsN;


unsigned long
bytesXSector,sectoresXCluster,entradasRaiz,sectoresReservados,sectoresXFAT,num
eroFATs;
unsigned long offsetParticion,offsetFAT,offsetRaiz,offsetData,dirActual;
unsigned short masArchivos;
void getCurDir();
/*
```

```
-------------------------------------------------------------------------------
-
F U N C I O N E S   G E N É R I C A S   D E   F A T
-------------------------------------------------------------------------------
-

*/
void leeCluster(unsigned long numClusterAbs)
{
    unsigned char temp[6];
    unsigned long cosa;
    int c,d;
    unsigned long direccion;
    direccion = numClusterAbs*maxBuffAdd;
    cosa = 256;
    temp[0] = 0x90;
    temp[1] = (direccion)%cosa;
    temp[2] = (direccion/(cosa))%cosa;
    temp[3] = (direccion/(cosa*cosa))%cosa;
    temp[4] = (direccion/(cosa*cosa*cosa))%cosa;
    temp[5] = 0x01;
    for(c=0;c<6;c++) serDputc(temp[c]);
    bloqueAbsN = numClusterAbs;
    for(c=0;c<maxBuffAdd;c++)
    {
      d = -1;
      while(d == -1){
         d = serDgetc();
      }
       bloqueAbs[c] = (unsigned char)d;
      //printf("%d)%x   ",c,bloqueAbs[c]);
    }
    return;
}
unsigned char leeByte(unsigned long direccion)
{
       //printf("Direccion %lx @ bloque %d. ",direccion,direccion/maxBuffAdd);

       if(direccion/maxBuffAdd==bloqueAbsN){
       //printf("Actual = %d leo offset %d",bloqueAbsN,(direccion-
bloqueAbsN*maxBuffAdd));
       return bloqueAbs[(direccion-bloqueAbsN*maxBuffAdd)];
   }
   else
   {
       leeCluster(direccion/maxBuffAdd);
      //printf("AHORA Actual = %d leo offset %d",bloqueAbsN,(direccion-
bloqueAbsN*maxBuffAdd));
      return bloqueAbs[(direccion-bloqueAbsN*maxBuffAdd)];
   }
}
unsigned char leeByteSC(unsigned long direccion)
{
    unsigned char temp[6];
    unsigned int temp2,c;
    unsigned long cosa;
    cosa = 256;
    temp2 = -1;
    temp[0] = 0x70;
    temp[1] = (direccion)%cosa;
    temp[2] = (direccion/(cosa))%cosa;
    temp[3] = (direccion/(cosa*cosa))%cosa;
    temp[4] = (direccion/(cosa*cosa*cosa))%cosa;
    temp[5] = 0x01;
    //if(direccion>=256) printf("%lu:
%x,%x,%x,%x\n",direccion,temp[1],temp[2],temp[3],temp[4]);
    for(c=0;c<6;c++) serDputc(temp[c]);
```

```
   while(temp2 == -1){
       temp2 = serDgetc();
   }
   //printf("Direccion[%x]Recibí %x\n",direccion,temp2);
       return (unsigned char) temp2;
}

unsigned int leeInt(unsigned long direccion)
{
//    printf("(Leo %d en %d y %d en %d)",
       return (unsigned int)(leeByte(direccion))+(unsigned
int)(256*leeByte(direccion+1));
}

unsigned int leeIntSC(unsigned long direccion)
{
//    printf("(Leo %d en %d y %d en %d)",
       return (unsigned int)(leeByteSC(direccion))+(unsigned
int)(256*leeByteSC(direccion+1));
}

void leeString(unsigned long direccion, unsigned long tamanio, char* destino){
       if(tamanio>=2048) tamanio = 2047;

if(direccion/maxBuffAdd==bloqueAbsN&&(direccion+tamanio)/maxBuffAdd==bloqueAbs
N){
       memcpy((void*)destino,(void*)(bloqueAbs+(direccion-
bloqueAbsN*maxBuffAdd)),tamanio);
   }
   else if(direccion/maxBuffAdd==bloqueAbsN){
       memcpy((void*)destino,(void*)(bloqueAbs+(direccion-
bloqueAbsN*maxBuffAdd)),2048-(direccion-bloqueAbsN*maxBuffAdd));
   }
   else
   {
       leeCluster(direccion/maxBuffAdd);
      memcpy((void*)destino,(void*)(bloqueAbs+(direccion-
bloqueAbsN*maxBuffAdd)),tamanio);

   }
   destino[tamanio]=0x00;
}

archivo* myfopen(archivo *arch, int offsetCluster)
{
    int i;
    printf("\nInicializando apertura de archivo %s",arch->nombre);
    if(offsetCluster==0) clustersArchivo[0] = arch->dirInicio;
    else clustersArchivo[0] = offsetCluster;
    printf("\n Clusters del archivo: %d,",clustersArchivo[0]);
    for(i=1;i<256;i++)
    {
        clustersArchivo[i] = leeInt(offsetFAT+2*clustersArchivo[i-1]);
        if(clustersArchivo[i]>=0xFFF8) break;
    }
    printf("\n");
    arch->abierto = 1;
    return arch;
}


void cd(archivo *arch)
{
    unsigned long ap,x;
    unsigned long a;
    char fecha[16];
    char tmp[4];
```

```
    unsigned int tmp2;
    printf("Llegamos aquí");
       if(arch->esDir>0)
    {
        //Sí es directorio, hacer el cambio
        if(arch->dirInicio==0){
            ap = offsetRaiz;
            dirActual = offsetRaiz;
            curDirArch = *arch;
        }
        else
        {
            ap = offsetData + (arch->dirInicio-2)*(unsigned
long)bytesXSector*(unsigned long)sectoresXCluster;
            dirActual = ap;
            curDirArch = *arch;
            printf("Como fue ROOTDIR, ap = %d o %lx\n\n",ap,ap);
        }

        for(x=0;x<128;x++)
        {
            if((leeByte(ap)==0x00)||((dirActual!=offsetRaiz)&&(ap-
dirActual>=bytesXSector*sectoresXCluster))){
                strcpy(archivos[x].nombre,"\0");
                break;
            }
            if((leeByte(ap)==0xE5)||(leeByte(ap+11)==0x0F)){
                //printf("[@%lx] Ignorando 1 elemento eliminado... \n",ap);
                ap+=32;
                x--;
                continue;
            }
            if(leeByte(ap+11)==0x08){
             ap+=32;
               x--;
             continue; //es VOLUME LABEL
            }


            //Entrada directorio
            archivos[x].dirEntry = ap;
            //Nombre
            leeString(ap,8,archivos[x].nombre);
            strcat(archivos[x].nombre,".");
            leeString(ap+8,3,tmp);
            strcat(archivos[x].nombre,tmp);
            archivos[x].nombre[12] = 0x00;
            //printf("[@%lx] Item %d: %s\n",ap,x,archivos[x].nombre);
            if(leeByte(ap+11)==((unsigned char)16))
            {
             printf(" (directorio) ");
                archivos[x].esDir = 1;
            }
            else
            {
                archivos[x].esDir = 0;
            }
            printf("%s\n",archivos[x].nombre);
            //¿Es directorio?

            //Cluster de inicio
            archivos[x].dirInicio = leeInt(ap+26);

            //Fecha (y hora...algún día)

            ltoa((leeInt(ap+0x18)&0x001F),tmp); //Día
            strcpy(fecha,tmp);
```

```
            ltoa((leeInt(ap+0x18)&0x01E0)>>5,tmp); //Mes
            strcat(fecha,"/");
            strcat(fecha,tmp);
            ltoa(((unsigned int)(leeInt(ap+0x18)&0xFE00))>>9,tmp); //Año
            strcat(fecha,"/");
            strcat(fecha,tmp);
            strcpy(archivos[x].fcreacion,fecha);

            //Tamaño
            archivos[x].tamanio = leeInt(ap+28)+leeInt(ap+30)*0xFFFF;

            //Ver si hay más
            if(leeByte(ap+32)==0)
            {
                strcpy(archivos[x+1].nombre,"\0");
                break;
            }
            else
            {
                //Caso normal
                ap+=32;
            }
        }
      }
   else printf("Este directorio no es archivo");
}

void inicializa()
{
      int c;
      //Empezamos con enviar los bytes de init a tarjeta
   for(c=0;c<6;c++) serDputc(initCmd[c]);
   while(serDgetc() != 0x01);
      if(leeByteSC(16)){
            bytesXSector = leeIntSC(11); //11 y 12
            sectoresXCluster = (unsigned int)leeByteSC(13); //sólo 13
            sectoresReservados = leeIntSC(14); //14 y 15
            //16 es number of FAT's
            entradasRaiz = leeIntSC(17); //17 y 18
            //19 y 20 son para #sectores, 21 para media descr.
            sectoresXFAT = leeIntSC(22);
            offsetParticion = 0;
            offsetFAT = offsetParticion + sectoresReservados*bytesXSector;
            offsetRaiz = offsetFAT+2*sectoresXFAT*bytesXSector;
            offsetData = offsetRaiz+entradasRaiz*32;
            dirActual = offsetRaiz;
            curDirArch.esDir = 1;
            strcpy(curDirArch.nombre,"/");
            curDirArch.dirInicio = 0;
            printf("Bytes por sector: %d\n",bytesXSector);
            printf("Sectores por cluster: %d\n",sectoresXCluster);
            printf("Sectores reservados: %d\n",sectoresReservados);
            printf("Entradas raíz máximas: %d\n",entradasRaiz);
            printf("Sectores por FAT: %d\n",sectoresXFAT);
      printf("Offset Directorio Raíz: %lx\n",offsetRaiz);
            printf("Offset FAT1: %lx\n",offsetFAT);
      printf("Ahora vamos a cd /\n");
      cd(&curDirArch);
       }
       else{
      printf("Error en detección de unidad");
            exit(1);
       }
}


void dir()
```

```
{
    archivo *arch;
    int x;
    arch = &curDirArch;

    printf("Leyendo directorio %s\n",arch->nombre);
        //Despliegue
    for(x=0;x<128;x++)
    {
        if(archivos[x].nombre[0]=='\0') break;
        printf("[0x%x] ",archivos[x].dirEntry);
        printf("%d) %s %s %d bytes   (cluster
%d)\n",x,archivos[x].nombre,((archivos[x].esDir==1)?" <DIR> ":"
"),archivos[x].tamanio,archivos[x].dirInicio);
    }


}

int miShtmlHandler(HttpState* estado)
{
       sprintf(estado->buffer,"Esto es una prueba...");
}

int changeDir(HttpState* estado)
{
   int cdIndex;
   printf("Llegué a inicio de changeDir\n");
       strcpy(stateURL,(estado->url+8));
   cdIndex = atoi(stateURL);
   printf("Obtuve cdIndex = %d\n",cdIndex);
   strcpy(curDir,archivos[cdIndex].nombre);
       printf("Cambié directorio actual a %s\n",curDir);
   printf("StateURL = %s\n",stateURL);
   printf("redirectTo = %s\n",redirectTo);
   cd(&archivos[cdIndex]);
   getCurDir();
   cgi_redirectto(estado,redirectTo);
       return 0;
}

void getCurDir()
{
   int x,y;
   char siz[10];
   char id[3];
   char *ptrDirText;
   ptrDirText = dirText;
   //Ahora sí viene lo bueno
   strcpy(ptrDirText," ");
   for(x=0;x<(sizeof(archivos)/sizeof(archivos[0]));x++)
   {
       if(archivos[x].nombre[0]==0x00){

       break;
       }
       printf("entradas[%d]\n",x);
      printf("\n.nombre = %s",archivos[x].nombre);
      printf("\n.esDir = %d",archivos[x].esDir);
      printf("\n.fcreacion = %s",archivos[x].fcreacion);
      printf("\n.tamanio = %d",archivos[x].tamanio);
      utoa(archivos[x].tamanio,siz);

       if(archivos[x].esDir==1)
       {

          //Es directorio
```

```
            strcpy(tmp,"");
            strcpy(tmp,"<tr><td>(DIR)</td><td><a href='do.cgi?");
            utoa(x,id);
            strcat(tmp,id);
            strcat(tmp,"'>");
            strcat(tmp,archivos[x].nombre);
            strcat(tmp,"</a>\n</td><td>");
            strcat(tmp,archivos[x].fcreacion);
            strcat(tmp,"</td><td></td></tr>\n");
         }
         else
         {
         //No es directorio
            strcpy(tmp,"<tr><td></td><td>");
            strcat(tmp,archivos[x].nombre);
            strcat(tmp,"</td><td>");
            strcat(tmp,archivos[x].fcreacion);
            strcat(tmp,"<td>");
            strcat(tmp,siz);
            strcat(tmp,"</td></tr>\n");
         }
         strcat(dirText,tmp);
            printf("strlen(tmp) = %d\n", strlen(tmp));
            for(y=0;y<strlen(tmp);y++){
               printf("%d.%c ",y,tmp[y]);
            }
   }
   printf("\n");
   for(x=0;x<strlen(dirText);x++){
      printf("%c",dirText[x]);
   }
}


const HttpSpec http_flashspec[] =
{
   { HTTPSPEC_FILE, "/",                          index_html, NULL, 0, NULL,NULL},
   { HTTPSPEC_FILE ,"/index.shtml", index_html, NULL, 0, NULL,NULL),
   { HTTPSPEC_VARIABLE,"curDir", 0, curDir, PTR16, "%s", NULL},
   { HTTPSPEC_VARIABLE,"currentIP", 0, curIP, PTR16, "%s", NULL},
      { HTTPSPEC_VARIABLE,"redir", 0, redirectTo, PTR16, "%s", NULL},
      { HTTPSPEC_VARIABLE,"textoDir", 0, dirText, PTR16, "%s", NULL},
   { HTTPSPEC_FUNCTION, "/do.cgi",      0,changeDir,0,NULL,NULL},
   { HTTPSPEC_VARIABLE,"stateurl", 0, stateURL, PTR16, "%s", NULL},
};


void main()
{
      //Puerto serial
      int c;
   HttpState* stat;
      BitWrPortI(PBDDR,&PBDDRShadow,1,7);
   BitWrPortI(PBDR,&PBDRShadow,1,7);              // SLEEP#
   BitWrPortI(PDDDR,&PDDDRShadow,1,3);
   BitWrPortI(PDDR,&PDDRShadow,0,3);              // CTS#
   BitWrPortI(PDDDR,&PDDDRShadow,1,6);
   BitWrPortI(PDDR,&PDDRShadow,0,6);              // DCD#
   BitWrPortI(PDDDR,&PDDDRShadow,1,5);
   BitWrPortI(PDDR,&PDDRShadow,0,5);              // DSR#
   BitWrPortI(PDDDR,&PDDDRShadow,1,7);
   BitWrPortI(PDDR,&PDDRShadow,1,7);              // RI#
   BitWrPortI(PDDDR,&PDDDRShadow,0,4);            // DTR#
   BitWrPortI(PDDDR,&PDDDRShadow,0,2);            // RTS#
   BitWrPortI(PEDDR,&PEDDRShadow,0,1);            // RXF#
   BitWrPortI(PBDDR,&PBDDRShadow,0,6);            // TXE#
```

```
    serDopen(691200);

    //Inicializar HTTP
    sock_init();
    http_init();
        tcp_reserveport(80);
    ifconfig(IF_DEFAULT,IFG_IPADDR,&currIP,IFS_END);
    inet_ntoa(curIP, currIP);
    printf("Dirección actual %s\n",curIP);
    strcpy(redirectTo,"http://");
    strcat(redirectTo,curIP);
    strcat(redirectTo,"/index.shtml");
    strcpy(curDir,"/");

        //Inicialización de FAT16 y Secure Digital
    bloqueAbsN = 0;
    inicializa();
    dir();
    getCurDir(stat);
    while(1)
    {
        http_handler();
    }
}
```

7.4. Emulador de SD. Código fuente.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Data;
using System.Threading;
using System.IO;

namespace SDEmulator
{
	public struct codigosUSB
	{
		public const byte initCmd = 0x00;
		public const byte resetCmd = 0x40;
		public const byte readCmd = 0x70;
		public const byte readBlockCmd = 0x90;
		public const int baudRate = 691200;
	}
	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	public class sd : System.Windows.Forms.Form
	{
		[DllImport("AID.dll")]
		static extern uint FT_ListDevices();
		[DllImport("AID.dll")]
		static extern uint FT_Open();
		[DllImport("AID.dll")]
		static extern uint FT_Close();
		[DllImport("AID.dll")]
		static extern uint FT_Write([MarshalAs(UnmanagedType.LPArray)]
byte[] p_data,ulong size);
		[DllImport("AID.dll")]
		static extern uint FT_GetStatus(ref ulong rxsize, ref ulong
txsize);
		[DllImport("AID.dll")]
		static extern uint FT_SetBitMode(byte mask, byte enable);
```

```
            [DllImport("AID.dll")]
            static extern uint FT_Read([MarshalAs(UnmanagedType.LPArray)]
byte[] p_data,ulong size);
            [DllImport("AID.dll")]
            static extern uint FT_EE_Read(ref ushort vid,ref ushort pid, ref
ushort power);
            [DllImport("AID.dll")]
            static extern uint FT_EE_Program(ushort power);
            [DllImport("AID.dll")]
            static extern uint FT_EE_ProgramToDefault();
            [DllImport("AID.dll")]
            static extern uint KCAN_Send(uint channel, uint id, uint dlc,
[MarshalAs(UnmanagedType.LPArray)] byte[] p_data);
            [DllImport("AID.dll")]
            static extern uint KCAN_Receive(ref uint channel, ref uint id,
ref uint dlc, [MarshalAs(UnmanagedType.LPArray)] byte[] p_data);
            [DllImport("AID.dll")]
            static extern uint KCAN_Init(uint baudraute);
            [DllImport("AID.dll")]
            static extern uint FT_SetBaudRate(uint braute);
            [DllImport("AID.dll")]
            static extern uint FT_PurgeTX();
            [DllImport("AID.dll")]
            static extern uint FT_PurgeRX();


            private bool enviar=false;

            private System.Windows.Forms.GroupBox grpHistorial;
            private System.Windows.Forms.TextBox txtHistorial;
            private System.Windows.Forms.Button btnNum;
            private System.Windows.Forms.GroupBox groupBox1;
            private System.Windows.Forms.Label label1;
            private System.Windows.Forms.Label label2;
            private byte[] codTX = new byte[512];
            private byte[] codRX = new byte[6];
            private Thread threadRX;
            private Thread threadRetry;
            private Thread threadTX;
            private System.IO.FileStream archivo = new
FileStream("C:\\Documents and Settings\\Rafael Tello\\Mis
documentos\\flash2web\\miusb.whx",System.IO.FileMode.Open);
            private System.Windows.Forms.Label stat;
            long dirBase = 0;
            private System.Windows.Forms.ProgressBar barRX;
            private System.Windows.Forms.ProgressBar barTX;
            protected int bytesAEnviar = 1;
            /// <summary>
            /// Required designer variable.
            /// </summary>
            private System.ComponentModel.Container components = null;
            public void escuchaUSB()
            {
                  byte[] datos = new byte[6]{0,0,0,0,0,0};
                  uint largo = 0;
                  int i=0;
                  sd.FT_SetBaudRate(codigosUSB.baudRate);
                  sd.FT_PurgeRX();
                  sd.FT_PurgeTX();
                  while(true)
                  {
                        datos.Initialize();
                        this.bytesAEnviar = 1;
                        this.barTX.Value = 0;
                        this.barRX.Value = 0;
                        largo = sd.FT_Read(datos,1);
                        this.barRX.Value = 1;
```

```
                        this.txtHistorial.SelectionStart =
this.txtHistorial.Text.Length;
                        switch(((byte)datos[0]))
                        {
                              case codigosUSB.resetCmd:
                                    for(i=0;i<5;i++)
                                    {
                                          largo = FT_Read(datos,1);
                                          this.barRX.Value++;
                                    }
                                    if(this.archivo.CanRead)
                                    {
                                          this.codTX[0] = 0x01;

      this.txtHistorial.Text+=DateTime.Now.ToString()+": Recibí comando de
RESET\r\n";
                                          while(this.enviar);
                                          enviar = true;
                                          while(this.enviar);
                                          this.stat.Text = "Rabbit ONLINE
@ "+codigosUSB.baudRate+" bps :)";
                                    }
                                    break;
                              case codigosUSB.readBlockCmd:
                                    this.dirBase = 0;
                                    for(i=0;i<4;i++)
                                    {
                                          largo = FT_Read(datos,1);
                                          this.dirBase = this.dirBase +
datos[0]*(long)Math.Pow(256,i);
                                          this.barRX.Value++;
                                    }
                                    largo = FT_Read(datos,1);
                                    //Leemos AHORA 512 bytes

      archivo.Seek(this.dirBase,System.IO.SeekOrigin.Begin);
                                    archivo.Read(codTX,0,512);

      this.txtHistorial.Text+=DateTime.Now.ToString()+": LECTURA DE BLOQUE a
dirección "+dirBase+"\r\n";
                                    while(this.enviar);
                                    this.bytesAEnviar = 256;
                                    enviar = true;
                                    while(this.enviar);
                                    break;
                              case codigosUSB.readCmd:
                                    this.dirBase = 0;
                                    for(i=0;i<4;i++)
                                    {
                                          largo = FT_Read(datos,1);
                                          this.dirBase = this.dirBase +
datos[0]*(long)Math.Pow(256,i);
                                          this.barRX.Value++;
                                    }
                                    largo = FT_Read(datos,1);
                                    //El paso macabro

      archivo.Seek(this.dirBase,System.IO.SeekOrigin.Begin);
                                    int x;
                                    x = archivo.ReadByte();
                                    if(x>=0) this.codTX[0] = (byte)x;
                                    else throw new Exception("x es -1");

      this.txtHistorial.Text+=DateTime.Now.ToString()+": LECTURA a dirección
"+dirBase+"\r\n";
                                    while(this.enviar);
                                    enviar = true;
```

```
                                        while(this.enviar);
                                        break;
                                }
                        }
                }
                private void TransmiteAUSB()
                {
                        int i;
                        while(true)
                        {
                                byte[] codTX2 = {0};
                                while(!this.enviar);
                                this.barTX.Value = 0;
                                this.txtHistorial.Text+=DateTime.Now.ToString();
                                for(i=0;i<this.bytesAEnviar;i++)
                                {
                                        codTX2[0]=codTX[i];
                                        sd.FT_Write(codTX2,1);
                                        this.barTX.Value = i/this.bytesAEnviar;
                                        Thread.Sleep(1);
                                }
                                this.txtHistorial.Text+=": Envié "+i+" bytes...
\r\n";
                                this.bytesAEnviar = 1;
                                this.enviar = false;
                        }
                }

                private void reintentarConnect()
                {
                        uint x;
                        while(true)
                        {
                                if(FT_ListDevices()>0)
                                {
                                        x = FT_Open();
                                        if(x==0)
                                        {
                                                sd.FT_SetBaudRate(codigosUSB.baudRate);
                                                this.stat.Text = "USB Conectado";
                                                this.threadTX.Start();
                                                threadRX.Start();
                                                Thread.CurrentThread.Abort();
                                        }
                                        else
                                        {
                                                this.stat.Text = "Error al conectar:
"+x.ToString();
                                                Thread.Sleep(1000);
                                        }
                                }
                                else
                                {
                                        this.stat.Text = "No hay dispositivo
conectado";
                                        Thread.Sleep(1000);
                                }

                        }
                }
                public sd()
                {
                        //
                        // Required for Windows Form Designer support
                        //
                        InitializeComponent();
                        threadRX = new Thread(new ThreadStart(this.escuchaUSB));
```

```
            threadTX= new Thread(new ThreadStart(this.TransmiteAUSB));
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.grpHistorial = new System.Windows.Forms.GroupBox();
            this.btnNum = new System.Windows.Forms.Button();
            this.txtHistorial = new System.Windows.Forms.TextBox();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.label2 = new System.Windows.Forms.Label();
            this.label1 = new System.Windows.Forms.Label();
            this.barRX = new System.Windows.Forms.ProgressBar();
            this.barTX = new System.Windows.Forms.ProgressBar();
            this.stat = new System.Windows.Forms.Label();
            this.grpHistorial.SuspendLayout();
            this.groupBox1.SuspendLayout();
            this.SuspendLayout();
            //
            // grpHistorial
            //
            this.grpHistorial.Controls.Add(this.btnNum);
            this.grpHistorial.Controls.Add(this.txtHistorial);
            this.grpHistorial.Location = new System.Drawing.Point(8,
8);
            this.grpHistorial.Name = "grpHistorial";
            this.grpHistorial.Size = new System.Drawing.Size(464, 288);
            this.grpHistorial.TabIndex = 0;
            this.grpHistorial.TabStop = false;
            this.grpHistorial.Text = "Historial de comandos";
            //
            // btnNum
            //
            this.btnNum.Location = new System.Drawing.Point(400, 248);
            this.btnNum.Name = "btnNum";
            this.btnNum.Size = new System.Drawing.Size(56, 23);
            this.btnNum.TabIndex = 1;
            this.btnNum.Text = "Limpiar";
            this.btnNum.Click += new
System.EventHandler(this.btnNum_Click);
            //
            // txtHistorial
            //
            this.txtHistorial.Location = new System.Drawing.Point(8,
24);
            this.txtHistorial.Multiline = true;
            this.txtHistorial.Name = "txtHistorial";
```

```
                    this.txtHistorial.ScrollBars =
System.Windows.Forms.ScrollBars.Vertical;
                    this.txtHistorial.Size = new System.Drawing.Size(384, 248);
                    this.txtHistorial.TabIndex = 0;
                    this.txtHistorial.Text = "";
                    //
                    // groupBox1
                    //
                    this.groupBox1.Controls.Add(this.label2);
                    this.groupBox1.Controls.Add(this.label1);
                    this.groupBox1.Controls.Add(this.barRX);
                    this.groupBox1.Controls.Add(this.barTX);
                    this.groupBox1.Controls.Add(this.stat);
                    this.groupBox1.Location = new System.Drawing.Point(8, 296);
                    this.groupBox1.Name = "groupBox1";
                    this.groupBox1.Size = new System.Drawing.Size(464, 80);
                    this.groupBox1.TabIndex = 1;
                    this.groupBox1.TabStop = false;
                    this.groupBox1.Text = "Status de puerto USB";
                    //
                    // label2
                    //
                    this.label2.Location = new System.Drawing.Point(248, 48);
                    this.label2.Name = "label2";
                    this.label2.Size = new System.Drawing.Size(40, 16);
                    this.label2.TabIndex = 4;
                    this.label2.Text = "RX";
                    //
                    // label1
                    //
                    this.label1.Location = new System.Drawing.Point(248, 24);
                    this.label1.Name = "label1";
                    this.label1.Size = new System.Drawing.Size(40, 16);
                    this.label1.TabIndex = 3;
                    this.label1.Text = "TX";
                    //
                    // barRX
                    //
                    this.barRX.Location = new System.Drawing.Point(296, 40);
                    this.barRX.Maximum = 6;
                    this.barRX.Name = "barRX";
                    this.barRX.Size = new System.Drawing.Size(152, 23);
                    this.barRX.TabIndex = 2;
                    //
                    // barTX
                    //
                    this.barTX.Location = new System.Drawing.Point(296, 16);
                    this.barTX.Name = "barTX";
                    this.barTX.Size = new System.Drawing.Size(152, 23);
                    this.barTX.TabIndex = 1;
                    //
                    // stat
                    //
                    this.stat.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
                    this.stat.Font = new System.Drawing.Font("Tahoma", 14F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                    this.stat.ForeColor = System.Drawing.Color.Brown;
                    this.stat.Location = new System.Drawing.Point(8, 16);
                    this.stat.Name = "stat";
                    this.stat.Size = new System.Drawing.Size(216, 56);
                    this.stat.TabIndex = 0;
                    this.stat.Text = "Sin conexión";
                    //
                    // sd
                    //
                    this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
```

```
                this.ClientSize = new System.Drawing.Size(480, 381);
                this.Controls.Add(this.groupBox1);
                this.Controls.Add(this.grpHistorial);
                this.Name = "sd";
                this.Text = "Emulador de Secure Digital";
                this.Load += new System.EventHandler(this.Form1_Load);
                this.grpHistorial.ResumeLayout(false);
                this.groupBox1.ResumeLayout(false);
                this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
                Application.Run(new sd());
        }


        private void btnNum_Click(object sender, System.EventArgs e)
        {
                this.txtHistorial.Text = "";
        }

        private void Form1_Load(object sender, System.EventArgs e)
        {
                if(FT_ListDevices()>0)
                {
                        uint x = FT_Open();
                        if(x==0)
                        {
                                this.stat.Text = "Conectado a chip USB";
                                threadRX.Start();
                                this.threadTX.Start();
                        }
                        else
                        {
                                threadRetry = new Thread(new
ThreadStart(this.reintentarConnect));
                                threadRetry.Start();
                                this.stat.Text = "Error al conectar:
"+x.ToString();
                        }
                }
                else
                {
                        threadRetry = new Thread(new
ThreadStart(this.reintentarConnect));
                        threadRetry.Start();
                        this.stat.Text = "No hay dispositivo conectado";
                }
        }
    }
}
```