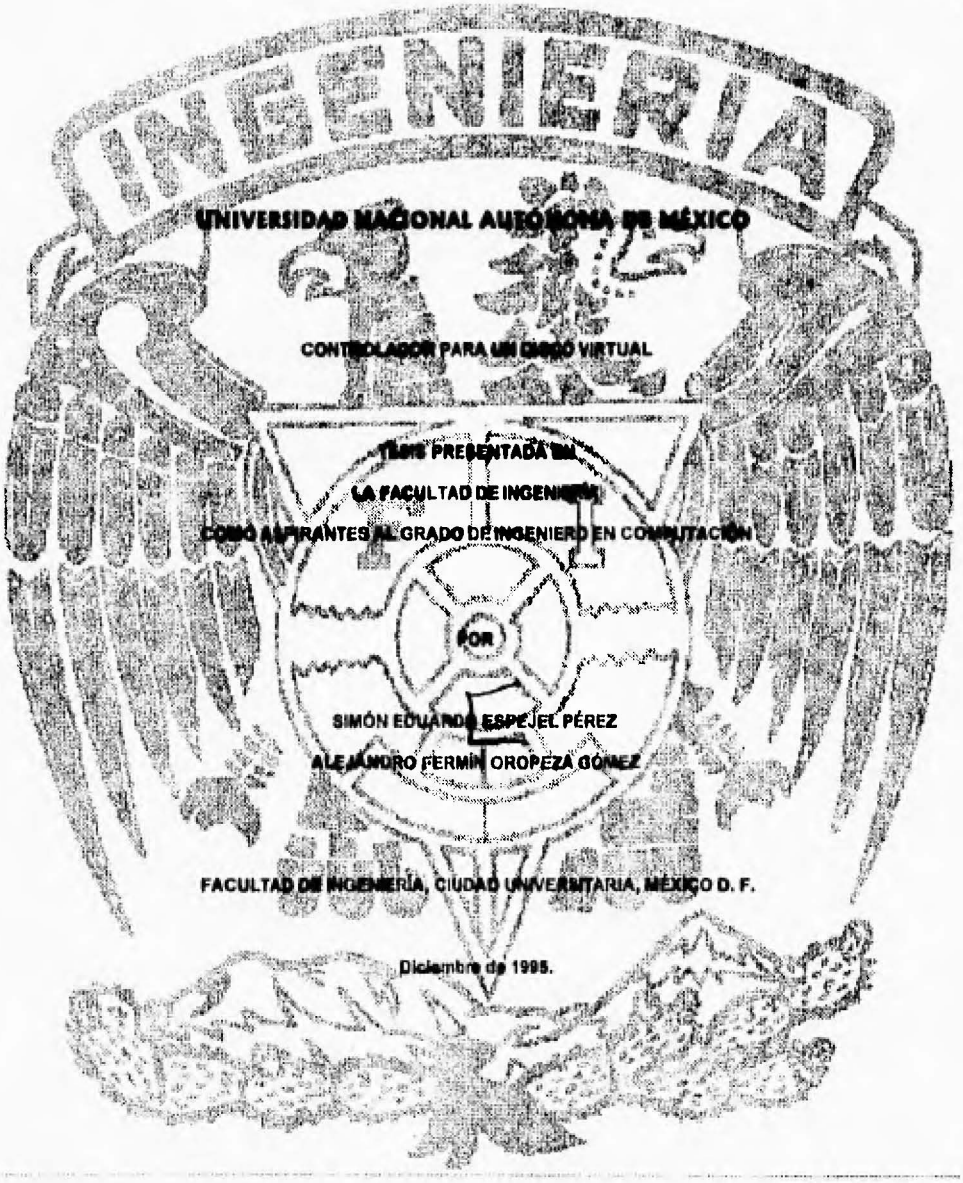
37
2ED




# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

## CONTROLADOR PARA UN DISCO VIRTUAL

TESIS PRESENTADA EN

LA FACULTAD DE INGENIERÍA

COMO ASPIRANTES AL GRADO DE INGENIERO EN COMPUTACIÓN

POR

SIMÓN EDUARDO ESPEJEL PÉREZ

ALEJANDRO FERMÍN OROPEZA GÓMEZ

FACULTAD DE INGENIERÍA, CIUDAD UNIVERSITARIA, MÉXICO D. F.

Diciembre de 1995.

FALLA DE ORIGEN

## *Dedicación.*

*A Ma. de Lourdes Gómez Macías.*

*Por mi existencia y formación profesional gracias a su cariño guía y apoyo, este trabajo simboliza mi gratitud por toda la responsable e invaluable ayuda que siempre me ha proporcionado.*

*A Lidia, Fermín y Filiberto por su gran apoyo.*

*A la memoria de Ofelia Macías.*

*A la enseñanza de mis maestros.*

*Alejandro Oropeza.*

*A la memoria de mi Madre.*
*Al respaldo de mis familiares y amigos.*
*A la enseñanza de mis Maestros.*

*Eduardo Espejel.*

# ÍNDICE

# 1. Introducción.

Acerca de este trabajo.

¿Qué es un Controlador de Dispositivos?

Breve Historia de los Controladores de Dispositivos.

## Lista de Tablas para el Capítulo 1.

# Capítulo 1
# Introducción

## Acerca de este trabajo.

El tema principal del presente trabajo son los Controladores de Dispositivos en el Sistema Operativo DOS.

El objetivo es mostrar un panorama general acerca del concepto de Controladores de dispositivos y su filosofía a través del tiempo y de diferentes ambientes de cómputo, centrándonos en el Sistema Operativo DOS de Microsoft e implementando una aplicación práctica que resuelva un problema real de manera sencilla para el usuario promedio. Su objetivo es mostrar la versatilidad y capacidades de un Controlador de dispositivos en DOS.

El trabajo está dividido en los siguientes capítulos:

1. Introducción.
- Se da una breve descripción del contenido de este trabajo.
- Se da una definición sencilla del concepto "Controlador de Dispositivos" y se da a conocer el contexto en que éstos se utilizan.
- Se relata someramente la historia de los Controladores de dispositivos.

2. Controladores de Dispositivos en algunos sistemas operativos.
- En este capítulo se describe brevemente la Filosofía que diversos sistemas operativos adoptan con respecto al Control de dispositivos periféricos. En algunos de ellos probablemente el concepto de controlador no existe como tal, pero el ejemplo es importante para mostrar la evolución que este concepto ha presentado a través del tiempo.

3. Controladores de Dispositivos en DOS.
- Se precisa el concepto "Controlador de Dispositivos" para el Sistema Operativo DOS, indicando de manera detallada y precisa cual es el formato y cuales las características que un Controlador de Dispositivos debe tener. Finalmente, se ofrecen algunas clasificaciones de los Controladores de Dispositivos.

4. Proyecto: Controlador para un disco virtual.
- Este capítulo detalla el ciclo de vida de la aplicación práctica que se eligió: Un controlador de discos virtuales que mapea al menos una unidad de disco a un archivo dentro de un Sistema de archivos de DOS. Éste es el capítulo más extenso del presente trabajo.

5. Conclusiones, Apéndices, Glosario, Bibliografía.
- La sección dedicada a las Conclusiones tiene como fin resumir los resultados obtenidos con el desarrollo de la aplicación práctica de este trabajo, así como revisar el cumplimiento de los objetivos planteados durante la fase de Análisis. Por otro lado, lista las posibles mejoras o correcciones que se puedan hacer al sistema en un trabajo futuro.

- Los apéndices proporcionados complementan el trabajo con información técnica importante para la comprensión del texto. Se incluyen los siguientes:
  1. Códigos fuente del Controlador así como de las utilerías desarrolladas. Se intentó conservar legibles y comentados los listados para facilitar su estudio.
  2. Debido a que estos conceptos resultan de central importancia para la comprensión del texto y de la aplicación, Se incluyó un apéndice que describe la estructura física genérica de un disco duro, así como el formato del Sistema de Archivos (Estructura lógica) que DOS aplica a un disco en DOS.
  3. Se describe la interfaz que BIOS proporciona para acceso a un disco duro, dado que se utiliza con frecuencia en el Controlador desarrollado.
  4. El apéndice que resume las estructuras de datos usadas por la interfaz de DOS a un Controlador de dispositivos se proporciona como referencia rápida a los Comandos aplicados en el Controlador.
- Se incluye un Glosario de los términos informáticos que se encuentran con mayor frecuencia durante la lectura del texto. No pretende ser totalizador, sino solo simplificar la comprensión de este trabajo.
- Se Listan las principales fuentes bibliográficas usadas como referencia durante el desarrollo del presente trabajo.

## ¿ Que es un controlador de dispositivos ?

Uno de los beneficios de la alta competitividad del mundo de la computación moderna es el surgimiento de una gran diversidad de opciones disponibles para configurar los centros de cómputo actuales. Así, es posible elegir de entre una gama prácticamente infinita de alternativas aquellas que se adapten mejor a nuestras necesidades, presupuesto, expectativas, aplicaciones, infraestructura instalada y hasta a nuestros gustos.

Paradójicamente, el surgimiento de esta diversidad ha planteado a su vez retos que vencer al mundo moderno. Uno de estos retos es el de lograr la convivencia de equipos que las más de las veces no serán homogéneos en cuanto a sus características de trabajo, velocidades, facilidad de uso, etcétera. Con el objetivo de suavizar estas diferencias y evitar el caos surgen continuamente estándares en todas las ramas de la informática (Comunicaciones, formatos gráficos, ambientes operativos, etcétera), estándares que se forman a partir del análisis de las alternativas comerciales que sean en ese instante las más avanzadas y/o populares del campo en cuestión. Sin embargo, nuevamente la competencia entre los proveedores de tecnología comienza a agregar nuevas características a dichos estándares hasta que nuevamente se requiere implantar un nuevo estándar que marque la pauta. Este ciclo es infinito y puede considerarse "la rueda del progreso" del mundo de la tecnología informática moderno. Obviamente aquellos estándares que son más sólidos y populares logran una vigencia mayor; Éste es el caso de los controladores de dispositivos en DOS.

Los controladores de dispositivos son el medio fundamental de comunicación de un equipo de cómputo moderno con el mundo exterior, equiparandose al cerebro animal en sus funciones de operación de los sentidos que proveen al ser vivo de información precisa acerca del mundo exterior. Siguiendo con la comparación, el "cerebro" de la computadora (En este caso el sistema operativo) se ve liberado así de lidiar con la gama tremenda de posibles dispositivos antes descrita. Por ejemplo, es gracias a los controladores de dispositivos que podemos conectar prácticamente cualquier dispositivo de Entrada/Salida (Monitores, Teclados, Unidades de discos, Impresoras, Digitalizadores, Ratones, Modems, Tarjetas de red, Lectores de código de barras, Unidades de cinta magnética, Lectores de CD-ROM, sensores de variables físicas, etcétera) a virtualmente cualquier computadora moderna. El sistema operativo solo define una interfaz estándar y hace llamadas a un controlador las más de las veces creado y provisto por el fabricante del dispositivo, y el controlador se encarga del resto. Desde este punto de vista los controladores de dispositivos cumplen una función importante al mantener un equipo de cómputo apto para convivir con los dispositivos que la tecnología vaya desarrollando.

## Breve Historia de los Controladores de Dispositivos

Para comenzar, es importante tener en mente que, al igual que el resto de los conceptos de la Computación, el desarrollo del concepto de Controlador de dispositivos está intimamente ligado al desarrollo que desde las primeras computadoras se ha dado a la industria del hardware. El desarrollo de nuevas tecnologías para la fabricación de circuitos electrónicos cada vez más pequeños, confiables y económicos en su costo de fabricación y en su consumo de potencia eléctrica (los equipos de cómputo actuales son miles de veces más pequeños, potentes, ahorradores y confiables que sus primeros predecesores) han permitido llevar sistemas de cómputo a cada vez más personas en el mundo. Esta tendencia surgió con las primeras computadoras y sigue evolucionando actualmente. De tal forma, se puede decir que el concepto de Controlador de dispositivos ha evolucionado a la par con las computadoras. Incluso, se podría clasificar en un caso para cada generación de computadoras.

El concepto de controlador de dispositivos, como ente con estructura y funciones bien diferenciadas del resto del sistema operativo, es relativamente joven. Sin embargo se pueden citar formas de control de dispositivos que a la larga dieron origen a la formación del concepto, conforme cobró importancia el contar con plataformas abiertas al control de múltiples productos de Hardware.

En los días de las primeras computadoras no se contaba con un esquema definido de manejo de dispositivos periféricos. El programador necesitaba unir al código de su aplicación código extra (por lo normal desarrollado por el mismo) que se encargaría exclusivamente de realizar las operaciones necesarias para controlar los dispositivos periféricos que su aplicación requiriera. Por supuesto,

los mecanismos de Entrada/Salida existentes eran mucho más sencillos que los actuales.

Con el paso del tiempo, las rutinas de Entrada/Salida usadas en los dispositivos periféricos de uso más común en aquel tiempo (Terminales tontas de Video, Lectores de tarjetas perforadas, Impresoras de trabajos en lote, etcétera) se volvieron de uso común y pasaron a ser del dominio público, dado que los Fabricantes de los equipos de cómputo y de los Sistemas operativos (En aquel tiempo siempre eran el mismo) comenzaron a incluirlas en librerías de código o en utilerías del Sistema operativo que eran invocadas por el programador o usuario en general. Éste fue el tiempo de los grandes sistemas de cómputo centralizados, grandes computadoras que ocupaban centros de trabajo con temperatura controlada y acceso restringido. La mayoría de las veces estos equipos convivían solamente con dispositivos que el mismo proveedor comercializaba y que estaban diseñados exclusivamente para su uso en esos equipos. Combinar una lectora de tarjetas (en aquellos tiempos común) y una computadora de diferentes fabricantes era en aquellos tiempos prácticamente impensable y un administrador de sistemas de un equipo específico tenía que dedicar gran parte de su tiempo a dominar cada aspecto del hardware y el software en que se desenvolvía, ignorando las demás tendencias informáticas que estuvieran surgiendo.

Conforme el desarrollo de hardware permitía que los sistemas de cómputo se volvieran más pequeños y económicos otra tendencia surgió, la de modificar el núcleo del Sistema operativo aumentando o modificando código que permitiera al usuario utilizar un dispositivo determinado. Esta operación representó un gran avance pues daba al usuario independencia de los fabricantes de dispositivos y facilitaba así el utilizar dispositivos de Entrada/Salida con computadoras que no provenían del mismo fabricante. Sin embargo, dicha proceso no era sencillo de realizar para el usuario común debido a que implicaba la modificación del Sistema Operativo y requería de una buena cantidad de conocimientos acerca del dispositivo en particular y del ambiente operativo en cuestión.
Alrededor de este tiempo toma forma el concepto concreto de controlador de dispositivos como tal.

Con el advenimiento de Sistemas Operativos más pequeños se adoptó un punto de vista más flexible y modular cuyo objetivo es permitir al usuario agregar fácilmente controladores de dispositivos que se encuentran en segmentos de código que están separados a su ambiente de trabajo sin afectar el código del Sistema Operativo. Tal es el caso de DOS, el sistema operativo que se tomó como plataforma para la aplicación práctica presentada en este trabajo.

La evolución actual de las plataformas de hardware y software parece tender hacias sistemas de cómputo con las siguientes capacidades:
1. Compactos, económicos, ligeros.

2. **Independientes.**
3. **Con interfaces (probablemente orientadas a objetos) amigables al usuario, pero eficientes.**
4. **Abiertos, con amplias capacidades de integración prácticamente transparente (En hardware y software) a otros sistemas no necesariamente iguales. Ésto puede dar origen a redes de equipos de cómputo no homogéneos, con usuarios conectados a través de medios de comunicación de naturalezas múltiples (linea telefónica, enlace satelital, fibra óptica, etcétera) utilizando computadoras de diferentes fabricantes y con sistemas operativos totalmente distintos entre si.**

**Ejemplos de ambientes que tratan de establecer esta tendencia (Aunque aún precozmente) son Windows NT, OS/2, NEXT, Ambientes integradores de Redes como Banyan Vines, UNIX, etcétera.**

**Dado el panorama presentado para esta última etapa, parece ser que la tendencia futura en controladores de dispositivos se dirige hacia la adición de una capa extra (conformada por el controlador) entre el hardware del sistema y una interfaz estándar al usuario, el cual usaría esta interfaz sin preocuparse por la plataforma de hardware que esté manejando. Esta situación se explica mejor en la sección dedicada a OS/2 del siguiente capitulo.**

**En resúmen, se observa que el concepto de Controlador de dispositivos ha evolucionado de la nada hasta ser considerado como un ente con atributos e interfaces bien definidos. Parece ser que esta tendencia continuará y un Controlador de dispositivos pasará a ser un agente inteligente al cual le indicaremos lo que deseamos y que se encargará de satisfacer nuestras peticiones sin importar los mecanismos que deba utilizar para conseguirlo.**

**Para concluir, la tabla 1.1 resume esta evolución y da ejemplos para cada renglón.**

| Etapa | Periodo | Descripción | Ejemplos |
|---|---|---|---|
| 1 | - 60's | Adición de control por cada desarrollador | Primeras computadoras |
| 2 | 60's - 70's | Librerías de rutinas para lograr control | - IBM 360 |
| 3 | 70's - 80's | Modificación de Kernel del Sistema Operativo para lograr control | - CP/M<br>- UNIX |
| 4 | 80's - 90's | Adición de Módulos de Control referenciados por el Sistema Operativo para lograr el control | - DOS |
| 5 | 90's - | Inserción de capa de controladores para proporcionar una plataforma estándar | - OS/2<br>- Windows NT<br>- Macintosh |

Tabla 1.1. Evolución de los Controladores de dispositivos.

# 2. Controladores de Dispositivos en algunos Sistemas Operativos.

Caso CP / M.

Caso UNIX.

Caso Apple Macintosh.

Caso OS/2.

## Lista de Tablas para el Capítulo 2.



## Lista de Figuras para el Capítulo 2.

# Capítulo 2
# Controladores de Dispositivos en algunos Sistemas Operativos

**En este capítulo se describe brevemente la forma en que algunos de los Sistemas operativos que existen han resuelto el problema de controlar diferentes dispositivos periféricos. Trata de mostrar la evolución del concepto, haciendo énfasis en lo que respecta a las técnicas de más reciente uso.**

## Caso CP/M.

**CP/M (Control Program for Microcomputers) fue uno de los primeros Sistemas operativos especialmente diseñados para micro computadoras, al principio de 8 bits. CP/M fue desarrollado por Gary Kildall en 1974 mientras trabajaba para MAA (Microcomputer Applications Associates, Compañía que posteriormente se convertiría en Digital Research) para apoyar al compilador residente de PL/M, un lenguaje desarrollado por el mismo en 1972 para Intel.**
**CP/M comenzó a ser distribuido comercialmente en 1975 y poco después ya era uno de los Sistemas operativos para micro computadoras más difundidos, gracias a características como el acceso de gran eficiencia a un sistema de archivos en disco flexible, confiabilidad, potencia, etcétera.**

**Una característica clave que permitió a CP/M su propagación a través de buena cantidad de plataformas de cómputo personal de 8 bits sin perder su esencia original y que es de importancia resaltar para este trabajo fue la división física del Sistema operativo en tres partes: el CCP (Console commands processor - Procesador de comandos de consola), el BDOS (Basic disk opeting system-Sistema Operativo de disco básico) y el BIOS (Basic Input/output system-Sistema Básico de Entrada/Salida).**
**Desde el punto de vista estratégico, CP/M se puede dividir en dos partes: Una variante y otra invariante.**

**Parte invariante - Contiene el Sistema Operativo de disco escrito en PL/M. Consta de 2 partes lógicas: CCP y BDOS.**
**Parte variante - Escrita en el lenguaje ensamblador nativo de la máquina en la cual se implementa CP/M, contiene los manipuladores de dispositivos de Entrada/Salida requeridos por la configuración local particular del hardware. Esta división posibilitó la conversión de CP/M en uno de los Sistemas Operativos más portátiles junto con UNIX. Esta parte contiene lo que en CP/M son los Controladores de Dispositivos y está conformada por el BIOS.**

**La parte que nos interesa en este trabajo es la variante (o BIOS), pues determina la forma en que CP/M convive con los dispositivos de Entrada/Salida y su estudio ayuda a comprender esta convivencia.**

**La parte que nos interesa en este trabajo es la variante (o BIOS), pues determina la forma en que CP/M convive con los dispositivos de Entrada/Salida y su estudio ayuda a comprender esta convivencia.**
**CP/M ve a los dispositivos como Lógicos o Físicos. Un dispositivo Físico es un dispositivo real (Hardware) capaz de realizar funciones de Entrada/Salida. En cambio, un dispositivo lógico es un artefacto provisto por el Sistema Operativo para hacer más favorable la interfaz al usuario; en realidad no existe. Los usuarios realizan peticiones de Entrada/Salida sobre dispositivos lógicos, de tal forma que la asignación de dispositivos lógicos a dispositivos físicos es lo que permite satisfacer dichas peticiones.**
**Los dispositivos lógicos estándar en CP/M son:**

- **CON: Designa un dispositivo de baja velocidad para comunicar al usuario con CP/M. Se asocian a el tres controladores de dispositivos lógicos:**

1. **CONIN: - Introduce un caracter a la vez desde el dispositivo de entrada estándar (Casi siempre la Consola).**
2. **CONOUT: - Envia un caracter a la vez al dispositivo de salida estándar (Casi siempre la Consola).**
3. **CONST: - Examina el dispositivo para determinar su condición y saber si hay o no algún caracter en espera de ser usado.**

- **RDR: Designa el dispositivo lector lógico utilizado para entradas de los dispositivos de almacenamiento masivo. Su manejador lógico se denomina READER.**
- **PUN: Designa el dispositivo lógico usado para realizar salidas a dispositivos de almacenamiento masivo. Su controlador lógico se denomina PUNCH. Llamado así por referencia a los perforadores de tarjetas.**
- **LST: Designa el dispositivo lógico para impresión de listados. Aunque normalmente dirige la salida hacia una impresora, también puede hacerlo hacia un dispositivo de almacenamiento masivo distinto. Su manejador lógico se denomina LIST.**

**Los dispositivos físicos estándar en CP/M se muestran en la tabla 2.1:**

| Nombre | Descripción |
|---|---|
| TTY: | Dispositivo de Teletipo |
| CRT: | Terminal de video/teclado |
| BAT: | Modo lote (Entrada RDR / Salida LST) |
| UC1: | Consola definida por el usuario |
| PTR: | Dispositivo de lectura de alta velocidad |
| UR1: | Lectora #1 definida por el usuario |
| UR2: | Lectora #2 definida por el usuario |
| PTP: | Dsipositivo perforador de alta velocidad |
| UP1: | Perforadora #1 definida por el usuario |
| UP2: | Perforadora #1 definida por el usuario |
| LPT: | Dispositivo de impresora de líneas |
| UL1: | Dispositivo de listado definido por el usuario |

Tabla 2.1. Dispositivos físicos estándar en CP/M.

**La parte que nos interesa en este trabajo es la varianta (o BIOS), pues determina la forma en que CP/M convive con los dispositivos de Entrada/Salida y su estudio ayuda a comprender esta convivencia.**
**CP/M ve a los dispositivos como Lógicos o Físicos. Un dispositivo Físico es un dispositivo real (Hardware) capaz de realizar funciones de Entrada/Salida. En cambio, un dispositivo lógico es un artefacto provisto por el Sistema Operativo para hacer más favorable la interfaz al usuario; en realidad no existe. Los usuarios realizan peticiones de Entrada/Salida sobre dispositivos lógicos, de tal forma que la asignación de dispositivos lógicos a dispositivos físicos es lo que permite satisfacer dichas peticiones.**
**Los dispositivos lógicos estándar an CP/M son:**

- **CON: Designa un dispositivo de baja velocidad para comunicar al usuario con CP/M. Se asocian a el tres controladores de dispositivos lógicos:**

1. **CONIN: - Introduce un caracter a la vez desde el dispositivo de entrada estándar (Casi siempre la Consola).**
2. **CONOUT: - Envia un caracter a la vez al dispositivo de salida estándar (Casi siampre la Consola).**
3. **CONST: - Examina el dispositivo para determinar su condición y saber si hay o no algún caractar en espera de ser usado.**

- **RDR: Designa el dispositivo lector lógico utilizado para entradas de los dispositivos da almacenamiento masivo. Su manejador lógico se denomina READER.**
- **PUN: Designa el dispositivo lógico usado para realizar salidas a dispositivos de almacenamiento masivo. Su controlador lógico se denomina PUNCH. Llamado asi por referencia a los perforadores de tarjetas.**
- **LST: Designa el dispositivo lógico para impresión de listados. Aunque normalmente dirige la salida hacia una impresora, también puede hacerlo hacia un dispositivo de almacenamiento masivo distinto. Su manajador lógico se denomina LIST.**

**Los dispositivos físicos estándar en CP/M se muestran en la tabla 2.1:**

| Nombre | Descripción |
|---|---|
| TTY: | Dispositivo de Teletipo |
| CRT: | Terminal de video/teclado |
| BAT: | Modo lote (Entrada RDR / Salida LST) |
| UC1: | Consola definida por el usuario |
| PTR: | Dispositivo de lectura de alta velocidad |
| UR1: | Lectora #1 definida por el usuario |
| UR2: | Lectora #2 definida por el usuario |
| PTP: | Dsipositivo perforador de alta velocidad |
| UP1: | Perforadora #1 definida por el usuario |
| UP2: | Perforadora #1 definida por el usuario |
| LPT: | Dispositivo de impresora de líneas |
| UL1: | Dispositivo de listado definido por el usuario |

Tabla 2.1. Dispositivos físicos estándar en CP/M.

Para terminar es conveniente señalar que, como se mencionó en el capítulo de "historia de los controladores de dispositivos", algunos sistemas operativos ofrecían la opción de permitir al usuario modificar su código para así adaptarse a nuevos dispositivos. Éste es el caso de CP/M, el cual permite al usuario modificar el BIOS del sistema para agregar control de nuevos dispositivos. Sin embargo, éste es un proceso un tanto complicado que implica el riesgo de dañar el BIOS de manera permanente.

## Caso UNIX.

En 1969 Bell Laboratories (en Estados Unidos) se retira del proyecto de desarrollo del Sistema operativo Multics y algunos miembros del grupo de investigación de Bell, dirigidos por Ken Thompson, comenzaron a trabajar en un Sistema Operativo mucho menos ambicioso escrito en lenguaje ensamblador. Este hecho marca el comienzo de la existencia de un Sistema operativo para minicomputadoras. Después de haber sido desarrollada, la primera versión de éste vio la luz por primera vez en 1970 corriendo en una máquina DEC PDP-7 (De la empresa Digital Equipment Corporation). Uno de los miembros del grupo, Brian Kernighan, bautizó en ese año al sistema como "UNIX", haciendo un juego de palabras con el antes mencionado Sistema operativo "MULTICS".

En 1973 el investigador Dennis Ritchie se unió el equipo original y colaboró en la reescritura del código fuante de UNIX en el recientemente creado lenguaje "C". Este fue uno de los hechos gracias a los cuales UNIX pasó de ser un Sistema operativo dedicado a aplicaciones y usuarios predominantemente científicos y técnicos a ser adoptado por los ámbitos administrativos e industriales a gran escala. Actualmente encabeza una de las tendencias clave en cuanto a la adopción de Sistemas operativos abiertos, y existen pocas plataformas de hardware que no puedan correr con alguna versión del Sistema Operativo UNIX, sin importar el fabricante o si la computadora es micro, mini o mainframe.

En lo que respecta al control de dispositivos, UNIX probablemente fue el primero en adoptar una metodología estricta encaminada a la fácil adaptabilidad a nuevos dispositivos, la cual entre otras cosas define los conceptos de dispositivo de caracter o de bloque, los de comandos de sistema operativo a controlador, y define la estructura y características indispensables que debe poseer un controlador agregado por terceros que deseen adaptar un nuevo dispositivo a UNIX. Ésta fue una de las razones que impulsaron a los diseñadores de las nuevas versiones de DOS a adoptar para éste una Filosofía semejante para el control de nuevos dispositivos, lo cual se manifiesta en las similitudes entre las definiciones de estructuras de los Controladores de dispositivos en ambos Sistemas Operativos.

Sin embargo, un punto a considerar es que la metodología seguida por UNIX es, al igual que en CP/M, la de modificar el código del Sistema Operativo. Es decir, en lo

**Sin embargo, un punto a considerar es que la metodología seguida por UNIX es, al igual que en CP/M, la de modificar el código del Sistema Operativo. Es decir, en lo que respecta al control de dispositivos, UNIX al igual que CP/M pertenece a la tercera generación de sistemas operativos (vease la sección llamada "Breve historia de los Controladores de Dispositivos"). Mientras en CP/M se modifica parte del BIOS, en UNIX se modifica el Kernel del Sistema Operativo. En otras palabras, los Controladores pasan a formar parte del Sistema Operativo. Ésta es una característica común a todos los Sistemas Operativos de aquella época[1] .**

**La figura 2.1 ilustra el concepto:**

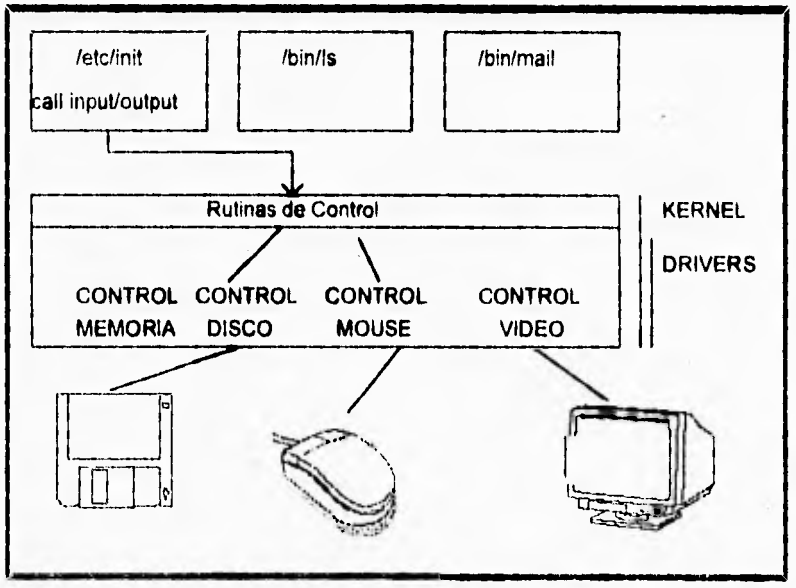



Figura 2.1. Control de dispositivos en UNIX

**Puede decirse que para UNIX un Controlador de Dispositivos se define como una colección de rutinas de software que forman parte del Sistema Operativo y que permiten al núcleo de UNIX y a programas de aplicación comunicarse con**

[1] A diferencia de CP/M, UNIX es un Sistema Operativo aún vigente y en evolución. Existen versiones para los más variadoas equipos, desde computadoras personales hasta SuperComputadoras. Debido a éste dinamismo los diseñadores de los nuevos sistemas UNIX ya han provisto mecanismos para aceptar de manera más eficiente y fácil nuevos dispositivos, sin romper con la Filosofía del Sistema operativo para el manejo de éstos.

concepción una impresora puede considerarse como un dispositivo de flujo de bytes de solo escritura, un teclado como un dispositivo de solo lectura, etcétera. Los dispositivos con que un sistema UNIX cuenta son visibles desde el sistema de archivos, dentro del directorio /dev. Los archivos dentro de este directorio se conocen como archivos "especiales" y pertenecen al superusuario. Todas las operaciones que se permiten con un archivo común y corriente son permitidas sobre un archivo dispositivo (Obviamente ciertas operaciones no tienen sentido, como mandar datos hacia un teclado). La única diferencia consiste en cual manejador de dispositivo será invocado al realizar una operación sobre algún archivo.
Como se pueda ver esta descripción concuerda en gran parte con la filosofía seguida por DOS y confirma que UNIX fue una de las fuentes de inspiración de los creadores de DOS.

## Caso Apple Macintosh.

Apple fue fundada en 1976 por Steve Jobs y Steve Wozniak. El primar producto de la nueva compañía fue la Apple II. Posteriormente, en 1984 introduce su computador personal Macintosh, que estaría destinado a marcar la pauta de una nueva era en Sistemas operativos gracias a innovaciones tales como el uso del ratón, una interfaz a usuario totalmente gráfica, uso de múltiples tipos y tamaños de letras, etcétera. Al igual que el resto del Sistema Operativo, El enfoque adoptado para el control de dispositivos en Macintosh fue bastante original. Su estructura es la siguiente:
Los dispositivos en Macintosh también se dividen en Dispositivos de caracter y Dispositivos de bloque. Los dispositivos de caracter (Como impresoras o puertos de comunicaciones) se consideran como dispositivos de acceso secuencial, mientras que los dispositivos de bloque (como unidades de disco duro/flexible) se consideran como dispositivos de acceso directo, además de realizar las operaciones de lectura/escritura siempre en grupos grandes de bytes.
Bajo condiciones normales de operación las aplicaciones se comunican con los dispositivos llamando a una entidad llamada Administrador de Dispositivos el cual puede ser llamado directamente por la aplicación o indirectamente por algún otro componente del sistema, como el Administrador de archivos o el Administrador de impresión. El Administrador de dispositivos no controla directamente ningún dispositivo en particular. Más bien, llama al manejador de dispositivo encargado específicamente de cada dispositivo. La figura 2.2 muestra este concepto.

Debido a que Macintosh es un Sistema Operativo propietario el equipo de cómputo relacionado no está sujeto a las generalizaciones a que un sistema más abierto (como una PC compatible con IBM) estaría sometido. Gracias a ésto es posible programar rutinas de importancia estratégica (como las de manejo de video) pare que resulten en código de gran eficiencia y guardarlas en la memoria ROM de la computadora, es decir, pasan a formar parte del firmware. Así, parte del Sistema

posible programar rutinas de importancia estratégica (como las de manejo de video) para que resulten en código de gran eficiencia y guardarlas en la memoria ROM de la computadora, es decir, pasan a formar parte del firmware. Así, parte del Sistema Operativo está previamente almacenado en la computadora. En el caso de los manejadores de dispositivos, también algunos de éstos fueron incluidos en ROM.
Los dispositivos que se encuentran en ROM son el de manejo de disco, el de sonido y el manejador en serie de ROM, mientras que algunos de los residentes en RAM son el de impresora, el manejador en serie de RAM, Appletalk y los accesorios de escritorio.

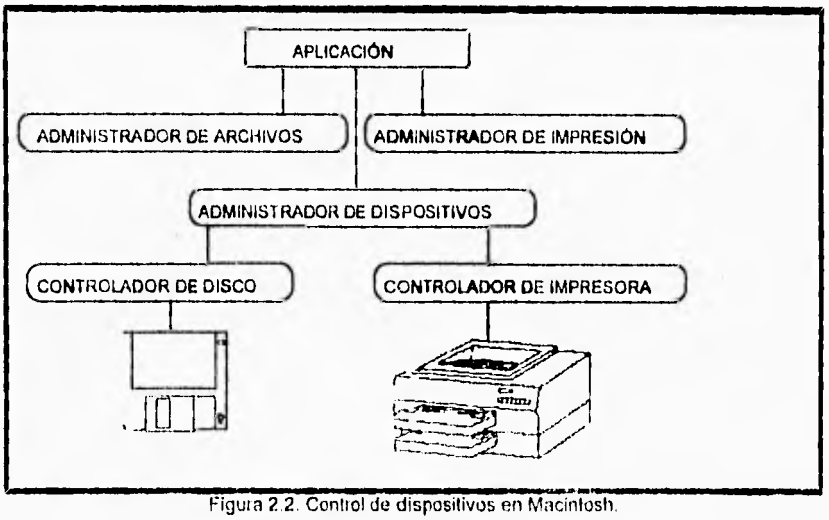



Figura 2.2. Control de dispositivos en Macintosh.

Los controladores de dispositivos pueden estar abiertos o cerrados. Mientras algunos (como el de disco y el de sonido) se abren automáticamente, otros tienen que ser abiertos por una llamada explícita. Como en otros sistemas operativos, las aplicaciones envían información de control a los controladores como: Tipo de protocolo, cerrar buffers, cambio de modo de operación, etcétera. Asimismo, el controlador retorna información de status a la aplicación. Cada vez que un manejador es abierto se agrega una entrada en una tabla de apuntadores al inicio del controlador y se inicializa una lista ligada de peticiones de comando al controlador recién abierto. Esta tabla se conoce con el nombre de Tabla Unitaria y se muestra un ejemplo de ella en la tabla 2.2.

Como se puede observar, el manejo de dispositivos en las computadoras Apple Macintosh es muy semejante al de otros sistemas operativos de la misma generación, Aunque Macintosh conserva un estilo propio que no demerita en nada su calidad. Ésta diferencia parte del hecho de que Macintosh desde su

fuertes. De ésta tendencia pueden surgir métodos de manejo de dispositivos más semejantes a los de computadoras PC pero compatibles con los actuales equipos Apple.

| Manejador | Unidad |
|---|---|
| Reservado. | 0 |
| de disco duro. | 1 |
| de impresora. | 2 |
| de sonido. | 3 |
| de disco. | 4 |
| En serie, puerto A. Input. | 5 |
| En serie, puerto A. Output. | 6 |
| En serie, puerto B. Input. | 7 |
| En serie, puerto B. Output. | 8 |
| de AppleTalk, MPP. | 9 |
| de AppleTalk, ATP. | 10 |
| Reservado. | 11 |
| Calculadora. | 12 |
| Reloj de Alarma. | 13 |
| Teclado. | 14 |
| Acertijo. | 15 |
| Cuaderno de Notas. | 16 |

Tabla 2.2. Ejemplo de Tabla unitaria en Macintosh.

## Caso OS/2.

En 1985 IBM y Microsoft firman un acuerdo para desarrollar un sistema operativo multitarea que trabajaría en el modo protegido del procesador 80286. El primer nombre del producto fue DOS versión 5.0, después CP/DOS y finalmente OS/2. Fue liberado algún tiempo después como la versión 1.0 y tuvo una interfaz orientada a caracteres.

Uno de los objetivos primarios establecidos durante el diseño de OS/2 fue el proveer una interfaz a usuario sin dependencias de dispositivos. En este sentido el caso OS/2 es un ejemplo claro de la tendencia futura en ambientes operativos. Antes de la existencia de estos ambientes el usuario o desarrollador de aplicaciones qua deseaban usar un dispositivo específico usaban un controlador proporcionado por el fabricante, lo cual implicaba que el desarrollador que había implementado una aplicación o el que la estaba usando debían conocer con precisión detalles importantes acerca del dispositivo que se estaba usando. Por ejemplo, no era igual programar una aplicación gráfica en un monitor CGA que en uno VGA, o imprimir una gráfica de barras en una impresora de matriz que en una láser. Algunas aplicaciones comerciales desarrollaban un conjunto de manejadores que luego

precisión detalles importantes acerca del dispositivo que se estaba usando. Por ejemplo, no era igual programar una aplicación gráfica en un monitor CGA que en uno VGA, o imprimir una gráfica de barras en una impresora de matriz que en una láser. Algunas aplicaciones comerciales desarrollaban un conjunto de manejadores que luego cambiaban cuando el dispositivo cambiaba. Sin embargo, Cuando otro desarrollador requería el mismo dispositivo tenía que desarrollar su propio controlador.

Con ambientes como OS/2, por otro lado, el proveedor desarrolla un solo controlador para cada dispositivo y después proporciona una interfez entre ese controlador y el usuario. Esta interfaz contiene un conjunto de comandos únicos para todo usuario. De esta forma, una aplicación puede solicitar a la interfaz gráfica que trace una línea desde el punto (0,0) al punto (0.5, 0.7) de una ventana y obtener la misma línea sin importar si el monitor es Hércules o SuperVGA. Esta misma filosofía se aplica a las otras interfaces, como las de Comunicaciones, impresión, intercambio de información entre procesos, etcétera.

La figura 2.3 ilustra este proceso para la interfaz a video.

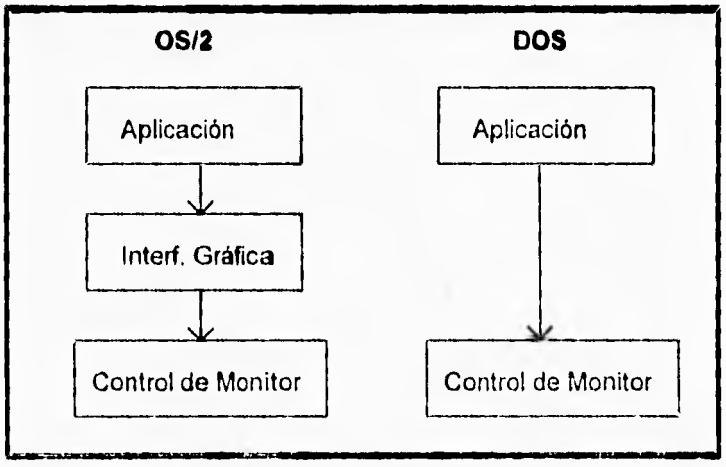



Figura 2.3. Capa de control intermedia en OS/2.

Otro de los objetivos planteados por los diseñadores de OS/2 es soportar entrada/salida rápida sobre dispositivos. En este sentido, el modelo tradicional de Controladores de Dispositivos de OS/2 puede ser muy lento. En previsión de esto último OS/2 permite a las aplicaciones obtener acceso directo a los puertos de E/S de un dispositivo y a su memoria (si la tiene).

toma el encabezado de la petición y lo forma en una lista ligada en espera de ser procesada. Otra similitud es la clasificación entre los controladores en tipo caracter y los de tipo bloque. La extensión es que mientras que los controladores de tipo caracter realizan su E/S de forma síncrona (Las llegadas se procesan por orden de llegada) los controladores tipo bloque lo hacen en forma asíncrona (El orden de atención no depende del orden de llegada). Por ejemplo, el controlador de disco recibe las solicitudes y las forma en una lista ligada donde son sorteadas atendiendo a criterios como su facilidad de acceso al disco o por su prioridad.
Finalmente, los controladores de dispositivos en OS/2 pueden operar en modo real o protegido, por lo cual se dice que trabajan de modo dual.

Otro aspecto digno de considerar es la existencia del "Ayudante de Dispositivos". Dado que los Controladores de Dispositivos en OS/2 se encuentran en el núcleo del sistema, no pueden llamar rutinas de la interfaz de OS/2 (Ésto se equipara con lo que ocurre en DOS, donde no se puede llamar una interrupción DOS desde un Controlador. Dicho sea de paso, en DOS no es una regla estricta, pero el desobedecerlo casi inevitablemente tendrá consecuencias impredecibles). Sin embargo, en OS/2 un controlador puede hacer llamadas a rutinas del "Ayudante de Dispositivos", el cual contiene servicios de bajo nivel para dar mayores capacidades a los controladores. Algunas de las categorías en que las rutinas del Ayudante se clasifican son:

- Administración del reloj del sistema.
- Administración de procesos.
- Administración de memoria.
- Servicios del sistema.
- Administración de supervisores.

Para terminar, se puede apreciar que la filosofía adoptada por OS/2 en el control de dispositivos representa una evolución a partir de la de DOS, refinando el concepto y haciendolo más poderoso. OS/2 representa una de las vertientes futuras en sistemas operativos y es un ejemplo típico de las técnicas de control de dispositivos que se usarán en los años venideros.

toma el encabezado de la petición y lo forma en una lista ligada en espera de ser procesada. Otra similitud es la clasificación entre los controladores en tipo caracter y los de tipo bloque. La extensión es que mientras que los controladores de tipo caracter realizan su E/S de forma síncrona (Las llegadas se procesan por orden de llegada) los controladores tipo bloque lo hacen en forma asíncrona (El orden de atención no depende del orden de llegada). Por ejemplo, el controlador de disco recibe las solicitudes y las forma en una lista ligada donde son sorteadas atendiendo a criterios como su facilidad de acceso al disco o por su prioridad.
Finalmente, los controladores de dispositivos en OS/2 pueden operar en modo real o protegido, por lo cual se dice que trabajan de modo dual.

Otro aspecto digno de considerar es la existencia del "Ayudante de Dispositivos". Dado que los Controladores de Dispositivos en OS/2 se encuentran en el núcleo del sistema, no pueden llamar rutinas de la interfaz de OS/2 (Ésto se equipara con lo que ocurre en DOS, donde no se puede llamar una interrupción DOS desde un Controlador. Dicho sea de paso, en DOS no es una regla estricta, pero el desobedecerlo casi invariablemente tendrá consecuencias impredecibles). Sin embargo, en OS/2 un controlador puede hacer llamadas a rutinas del "Ayudante de Dispositivos", el cual contiene servicios de bajo nivel para dar mayores capacidades a los controladores. Algunas de las categorías en que las rutinas del Ayudante se clasifican son:

- Administración del reloj del sistema.
- Administración de procesos.
- Administración de memoria.
- Servicios del sistema.
- Administración de supervisores.

Para terminar, se puede apreciar que la filosofía adoptada por OS/2 en el control de dispositivos representa una evolución a partir de la de DOS, refinando el concepto y haciendolo más poderoso. OS/2 representa una de las vertientes futuras en sistemas operativos y es un ejemplo típico de las técnicas de control de dispositivos que se usarán en los años venideros.

# 3. Controladores de Dispositivos en DOS.

Resúmen histórico del Sistema Operativo DOS.

Filosofía.

Clasificación.

Estructura.

Comunicación entre DOS y el Controlador.

Comandos enviados por DOS al Controlador.

## Lista de Tablas para el Capítulo 3.



## Lista de Figuras para el Capítulo 3.

# Capítulo 3
# Controladores de Dispositivos en DOS

## Resúmen histórico del Sistema operativo DOS

**En poco más de una década, el sistema operativo MS-DOS ha pasado de ser un simple cargador de programas a un sofisticado sistema operativo para computadoras basadas en la familia de procesadores 80X86 de Intel.**

**El progenitor de MS-DOS fue un Sistema Operativo de nombre 86-DOS, el cual fue escrito por Tim Patterson para Seattle Computers a mediados de 1980. En aquel tiempo, el Sistema Operativo CP/M-80 de Digital Research era el preferido para las microcomputadoras basadas en los procesadores 8080 y Z-80 de Intel y Zilog, respectivamente.**

**Originalmente, 86-DOS fue concebido para facilitar la migración de aplicaciones desde CP/M-80. Como consecuencia de ésto, ambos sistemas operativos resultaron ser muy semejantes interna y externamente.**

**En Octubre de 1980, IBM contactó a las mayores casas productoras de software en los Estados Unidos en busca de un Sistema Operativo para su nueva linea de computadoras. Microsoft pagó a Seattle Computers una cuota por los derechos para participar con 86-DOS y posteriormente, en Julio de 1981, lo compró, modificó y renombró como MS-DOS.**

**Para fines del 81, IBM estaba comenzando a vender sus microcomputadoras con MS-DOS y otros dos Sistemas Operativos alternos: CP/M-86 y P-System (de Digital Research y Softech, respectivamente). Finalmente, MS-DOS ganó la batalla y fue elegido por IBM como el Sistema Operativo para sus microcomputadoras. Desde ese momento, MS-DOS ha evolucionado hasta su versión 6.21.**

**Una de las actualizaciones mayores de MS-DOS, la versión 2.0, tomó numerosas caracteristicas del sistema operativo multitarea UNIX, por ejemplo:**

1. **Esquema de administración de archivos, lo cual incluyó:**
   - **Estructura jerárquica en su sistema de archivos.**
   - **Redirección de Entrada/Salida.**
2. **Uso de Redirecciones y Filtros de salida.**
3. **Metodologia para el Control de dispositivos, totalmente consistente con su filosofía de administración de archivos y con una interfaz de acceso bien delimitada.**

**De los puntos mencionados el último renglón reviste particular importancia para este trabajo, pues marcó la pauta para hacer de DOS un sistema operativo más abierto, amigable y extensible al uso de dispositivos de Entrada/Salida adicionales. Sin embargo, Microsoft agregó una característica más a esta metodologia: Controladores de dispositivos fácilmente instalables y desinstalables.**

## Filosofía.

Si bien comenzó sólo como una extensión más al Sistema operativo MS-DOS, la adición de la interfaz a dispositivos definida con la versión 2.0 (y concebida en gran parte a semejanza de la de UNIX) ha probado ser una de las razones fundamentales de la prolongada permanencia de MS-DOS en la preferencia de varios miles de usuarios.

Entre otras cosas los Controladores de dispositivos:

- Le dan a DOS la capacidad de mejorar o hasta reemplazar a los dispositivos tradicionales con otros mejorados. Ejemplos de ésto son:
  - Reemplazo del Controlador de video original, con programas como Ansi.sys.
  - Soporte a diferentes idiomas en su teclado (Keyb.com).
  - Uso de Discos en memoria RAM (Discos Virtuales, con Vdisk.sys).
  - Impresión en Background (Print.sys).

- Capacitan a MS-DOS para expandir sus capacidades conforme los avances tecnológicos lo vayan precisando. Le permiten por Ejemplo:
  - Accesar la parte alta de la memoria (Himem.sys).
  - Soportar memoria expandida formato LIM (Emm386.sys).
  - Soporte a dispositivos tipo ratón (Mouse.com).
  - Interfaz con unidades lectoras de CD-ROM (Mscdex.com).

Todos estos ejemplos son solo extensiones que Microsoft ha hecho a su producto para mantenerlo acorde con los avances tecnológicos que progresivamente han surgido. Sin embargo, oleadas de fabricantes siguen creando controladores para todo tipo de productos como Lectores de cinta, Unidades de discos ópticos, Pantallas sensibles al tacto, acceso a Redes de computadoras de diversos alcances, etcétera. Todo ésto habria sido muy difícil sin el uso del concepto de Controladores de dispositivos.

## Clasificación.

Dado que la filosofía de control de dispositivos de DOS fue tomada en muchos aspectos de UNIX, es de esperarse que DOS los clasifique de una forma semejante. Dicho sea de paso, esta filosofía ha sido adoptada por otros Sistemas operativos más recientes, como OS/2 o el de los equipos Apple. Esta clasificación atiende a la forma en que los dispositivos transfieren datos de y hacia la computadora y divide a los controladores de dispositivos en 2 tipos:

## De caracter.

**Los controladores de tipo caracter están diseñados para atender a dispositivos que fueron diseñados para realizar operaciones de entrada/salida de longitud variable. Es decir, con un controlador de este tipo es posible realizar transferencias desde uno y hasta cualquier número de caracteres. Ejemplos típicos de dispositivos atendidos por este tipo de controladores son los monitores, teclados, impresoras, redes, dispositivos apuntadores (como el ratón), modems, etcétera.**

## De bloque.

**Los controladores de tipo bloque manejan sus datos en grupos de caracteres y transfieren varios bytes a la vez en un solo bloque (comúnmente en múltiplos de 512). Algunos ejemplos de este tipo de Controladores son los Discos (de cualquier tipo) o las unidades de cinta. Las dispositivos características de esta clase de controladores son capaces de lograr altas tasas de transferencia de datos por unidad de tiempo. Por ejemplo, si en un disco duro realizaramos transferencias de tipo caracter el acceso a este dispositivo sufriría un grave deterioro pues para cuando el controlador transfiriera un caracter al disco éste ya habría revolucionado lo suficiente como para impedir la escritura del caracter contiguo. Éste incrementaría en mucho la cantidad de revoluciones y el tiempo requeridos para realizar la transferencia de un grupo de datos grande al disco.**

**Para ejemplificar, a continuación se listan los dispositivos estándar bajo DOS y se indica su tipo en base a esta clasificación:**

| Nombre | Tipo | Descripción |
|---|---|---|
| CON: | Caracter | Teclado/Pantalla |
| COM1: | Caracter | Puerto serial 1 |
| AUX: | Caracter | Igual a COM1 |
| COM2: | Caracter | Puerto serial 1 |
| LPT1: | Caracter | Puerto paralelo 1 |
| LPT2: | Caracter | Puerto paralelo 2 |
| LPT3: | Caracter | Puerto paralelo 3 |
| PRN: | Caracter | Idéntico a LPT1 |
| NUL: | Caracter | Dispositivo nulo |
| CLOCK$ | Caracter | Reloj en Software |
| A: | Bloque | Primera unidad de disquete |
| B: | Bloque | Segunda unidad de disquete |
| C: | Bloque | Disco Duro (Si existe) |

Tabla 3.1. Dispositivos estándar en DOS.

## Otras Clasificaciones.

**La clasificación anterior está basada en el tipo de transferencias que un dispositivo puede realizar. Sin embargo, una clasificación menos estricta puede basarse en el tipo de dispositivo que se esté controlando, lo cual ilustraría las variadas capacidades de un Controlador de dispositivos. Por ejemplo:**

**Controladores para nuevos dispositivos.**
**Como se comentó anteriormente, una de las capacidades clave de DOS es su capacidad de adaptación a nuevos dispositivos que con el tiempo vayan surgiendo. Esta capacidad se debe en gran parte a los Controladores de dispositivos. Cuando un nuevo dispositivo surge, el fabricante lo distribuye junto con el software que lo controla y así DOS es capaz de aprovecharlo. Ejemplos de ésto son nuevos discos ópticos reescribibles, lectores de CD-ROM, nuevas unidades de cinta, etcétera.**

**Controladores de reemplazo.**
**Son los usados cuando se desea modificar las características de un dispositivo cuyo controlador ya existía previamente. Un ejemplo de ésto es el controlador Ansi.sys, que reemplaza al dispositivo CON: estandar.**

**Controladores sin dispositivos.**
**Son aquellos que no controlan dispositivos de hardware reales, por lo cual son en ocasiones clasificados como "Controladores de dispositivos virtuales". La función de estos controladores es manipular de alguna forma un recurso existente para simular un dispositivo y lograr así una ventaja importante. Ejemplos de este tipo de dispositivos son el disco en RAM de DOS, una unidad asignada a una ruta dentro de un servidor en red, el dispositivo estándar de DOS llamado NUL: o incluso la aplicación desarrollada en este trabajo.**

## Estructura.

Descrito de manera conceptual, un programa Controlador de Dispositivos DOS consiste de cinco partes: encabezado del dispositivo (Device Header), áreas de almacenamiento de variables y rutinas locales, la rutina Estrategia, la rutina Interrupción, y las rutinas de procesamiento de comandos y de apoyo que el controlador en particular requiera.

Físicamente, Un Controlador de dispositivos DOS es muy semejante a un archivo ejecutable tipo .COM en cuanto a que es una imágen exacta de memoria, salvo

## Otras Clasificaciones.

La clasificación anterior está basada en el tipo de transferencias que un dispositivo puede realizar. Sin embargo, una clasificación menos estricta puede basarse en el tipo de dispositivo que se esté controlando, lo cual ilustraría las variadas capacidades de un Controlador de dispositivos. Por ejemplo:

**Controladores para nuevos dispositivos.**
Como se comentó anteriormente, una de las capacidades clave de DOS es su capacidad de adaptación a nuevos dispositivos que con el tiempo vayan surgiendo. Esta capacidad se debe en gran parte a los Controladores de dispositivos. Cuando un nuevo dispositivo surge, el fabricante lo distribuye junto con el software que lo controla y así DOS es capaz de aprovecharlo. Ejemplos de ésto son nuevos discos ópticos reescribibles, lectores de CD-ROM, nuevas unidades de cinta, etcétera.

**Controladores de reemplazo.**
Son los usados cuando se desea modificar las características de un dispositivo cuyo controlador ya existía previamente. Un ejemplo de ésto es el controlador Ansi.sys, que reemplaza al dispositivo CON: estandar.

**Controladores sin dispositivos.**
Son aquellos que no controlan dispositivos de hardware reales, por lo cual son en ocasiones clasificados como "Controladores de dispositivos virtuales". La función de estos controladores es manipular de alguna forma un recurso existente para simular un dispositivo y lograr así una ventaja importante. Ejemplos de este tipo de dispositivos son el disco en RAM de DOS, una unidad asignada a una ruta dentro de un servidor en red, el dispositivo estándar de DOS llamado NUL: o incluso la aplicación desarrollada en este trabajo.

## Estructura.

Descrito de manera conceptual, un programa Controlador de Dispositivos DOS consiste de cinco partes: encabezado del dispositivo (Device Header), áreas de almacenamiento de variables y rutinas locales, la rutina Estrategia, la rutina Interrupción, y las rutinas de procesamiento de comandos y de apoyo que el controlador en particular requiera.

Fisicamente, Un Controlador de dispositivos DOS es muy semejante a un archivo ejecutable tipo .COM en cuanto a que es una imágen exacta de memoria, salvo

por una diferencia: los programas .COM requieren iniciar en el byte 256 (100h) en memoria RAM, para que DOS pueda agregar un PSP (Program Segment Prefix-Prefijo de Segmentos para el programa) al inicio del programa en memoria, mientras los Controladores de dispositivos no tienen esta restricción. A cambio, cada Controlador de dispositivos deberá tener el encabezado antes mencionado, que es análogo al PSP de un programa .COM. Después del encabezado se encuentran variables y rutinas locales. A continuación vienen las rutinas Estrategia e Interrupción, indispensables para la operación de cualquier controlador de dispositivos. Finalmente, se encuentran las rutinas encargadas de procesar cada uno de los comandos que DOS pase a un Controlador de dispositivos.

El Encabezado del dispositivo, representado como una estructura en lenguaje C, consta de las siguientes partes:

```
typedef    struct device_header_struct {
      struct device_header_struct    far * sigdev ;
      unsigned int                        atributo ;
      void                                (* estrategia) (void);
      void                                (* interrupcion) (void);
      unsigned char                      nombre [8] ;
} deviceheader_t ;
```

Los significados de cada campo son los siguientes:

**sigdev.**
Si el archivo contiene más de un controlador, este campo contiene un apuntador al siguiente controlador dentro del archivo, o cero si es el último controlador. Cuando se carga en memoria el controlador este campo cambia su valor de acuerdo a la posición en memoria del siguiente controlador en la lista ligada de controladores manejada por DOS.

**atributo.**
Dos bytes que indican las características del Controlador y de cuyo valor dependerán gran parte de sus características y comportamiento. En realidad, el modificar la información contenida en esta palabra implica dar al controlador una personalidad totalmente distinta. Podría decirse que equivale a la huella digital del controlador. En la presentación de nuestro proyecto empezaremos por explicar la palabra de atributo que se dará al sistema y a partir de ésta se inferirán los comandos a implementar. Su formato es:

| Bit # | Descripción |
|---|---|
| 0 | 1-Es el dispositivo actual de entrada |
| 1 | 1-Es el dispositivo actual de salida |
| 2 | 1-Es el dispositivo NUL actual |
| 3 | 1-Es el dispositivo Clock actual |
| 4 | Especial |
| 5 | Reservado |
| 6 | Soporta ConTroL de E/S genérico |
| 7 | Soporta Queries de ConTroL de E/S genérico |
| 8 | Reservado |
| 9 | Reservado |
| 10 | Reservado |
| 11 | 1-Soporta medios removibles |
| 12 | Reservado |
| 13 | Dispositivo de bloque No-IBM |
| 14 | Soporta ConTroL de E/S |
| 15 | Dispositivo de: 0-Bloque. 1-Caracter |

Tabla 3.2. Campo de Atributo de un Controlador de dispositivos en DOS.

- **Los bytes definidos como reservados deberán ser puestos en Cero.**
- **Los primeros cuatro bytes indican si el controlador reemplaza a los dispositivos estándar Consola o Clock (Aún cuando el bit 2 esté en Uno ningún dispositivo puede reemplazar al dispositivo NUL. A excepción de éste, todos los dispositivos son reemplazables).**
- **El bit 4 se usa para indicar si el controlador soporta Entrada/Salida rápida por consola, la cual se programa modificando el código de la interrupción 29h. Ésta es una caracteristica que está más allá del contexto de este trabajo.**
- **El bit 6 se utiliza a partir de la versión de DOS 3.2 e indica si el controlador implementará el comando 19, definido más adelante (Vease la sección "Comandos enviados por DOS al Controlador" en el Comando Generic IOCTL).**
- **El bit 7 se usa a partir de la versión de DOS 5.0, e indica si el controlador soportará el comando 25, definido adelante (Vease la sección "Comandos enviados por DOS al Controlador" en el Comando IOCTL Query).**
- **El bit 11 define si el Controlador soportará los comandos 13, 14 y 15, definidos más adelante (Vease la sección "Comandos enviados por DOS al Controlador" en los Comandos Device Open, Device Close y Removable Media).**
- **El bit 13 indica si el formato del dispositivo de bloque es IBM o no, lo cual implica cambios en la forma en que DOS tratará al controlador.**
- **El bit 14 define si el Controlador soportará los comandos de control de Entrada/Salida, definidos más adelante (Vease la sección "Comandos enviados por DOS al Controlador" en los Comandos 3 y 12).**

- **El bit 15 es crucial pues define si el controlador es de tipo bloque o de tipo caracter y los demás bits de este campo pueden variar su significado dependiendo de éste.**

**La tabla 3.3 muestra de manera resumida qué bits de atributo, cuando se les asigna un valor Uno, provocarán que ciertos comandos sean enviados al controlador. Los comandos que no aparecen en la tabla serán invocados invariablemente, sin importar los atributos (Siempre y cuando sean invocables para el tipo de controlador en cuestión).**

**Estrategia e Interrupción.**
**Son campos que se interpretan como apuntadores a las rutinas del mismo nombre que se encuentran dentro del cuerpo del controlador y que ya han sido mencionadas con anterioridad. La utilidad de estas rutinas se explica en la siguiente sección.**

**nombre.**
**Si el controlador es de tipo caracter, este campo debe contener el nombre que el dispositivo tendrá. Como ejemplo están los controladores estándar NUL(dispositivo nulo), COM1(primer puerto serial), LPT1(primer puerto paralelo), etcétera. Si el Controlador es de tipo bloque, el primer byte de este campo tendrá un número indicando el total de dispositivos que el controlador manejará (tentativamente, pues la cantidad definitiva será determinada por el controlador mismo) y los restantes 7 bytes estarán vacios.**
**Las partes restantes del Controlador se describen a detalle en las siguientes secciones.**

| Comandos Enviados | Bits de atributo en Uno | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IOCTL Input | | A | | | | | | | | | | | | | | |
| IOCTL Output | | A | | | | | | | | | | | | | | |
| Device Open | | | | | A | | | | | | | | | | | |
| Device Close | | | | | A | | | | | | | | | | | |
| Removable | | | | | B | | | | | | | | | | | |
| Output Til Busy | | | C | | | | | | | | | | | | | |
| Generic IOCTL | | | | | | | | | | A | | | | | | B |
| Get Logical Dev | | | | | | | | | | B | | | | | | |
| Set Logical Dev | | | | | | | | | | B | | | | | | |
| IOCTL Query | | | | | | | | | A | | | | | | | |

**Dispositivos de: B:BLOQUE --- C:CARACTER --- A:AMBOS**

Tabla 3.3. Relación entre el campo de Atributo de un Controlador en DOS y los comandos que se invocan para el Controlador.

- El bit 15 es crucial pues define si el controlador es de tipo bloque o de tipo caracter y los demás bits de este campo pueden variar su significado dependiendo de éste.

La tabla 3.3 muestra de manera resumida qué bits de atributo, cuando se les asigna un valor Uno, provocarán que ciertos comandos sean enviados al controlador. Los comandos que no aparecen en la tabla serán invocados invariablemente, sin importar los atributos (Siempre y cuando sean invocables para el tipo de controlador en cuestión).

**Estrategia e Interrupción.**
Son campos que se interpretan como apuntadores a las rutinas del mismo nombre que se encuentran dentro del cuerpo del controlador y que ya han sido mencionadas con anterioridad. La utilidad de estas rutinas se explica en la siguiente sección.

**nombre.**
Si el controlador es de tipo caracter, este campo debe contener el nombre que el dispositivo tendrá. Como ejemplo están los controladores estándar NUL(dispositivo nulo), COM1(primer puerto serial), LPT1(primer puerto paralelo), etcétera. Si el Controlador es de tipo bloque, el primer byte de este campo tendrá un número indicando el total de dispositivos que el controlador manejará (tentativamente, pues la cantidad definitiva será determinada por el controlador mismo) y los restantes 7 bytes estarán vacíos.
Las partes restantes del Controlador se describen a detalle en las siguientes secciones.

| Comandos Enviados | Bits de atributo en Uno | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IOCTL Input | | A | | | | | | | | | | | | | | |
| IOCTL Output | | A | | | | | | | | | | | | | | |
| Device Open | | | | | A | | | | | | | | | | | |
| Device Close | | | | | A | | | | | | | | | | | |
| Removable | | | | | B | | | | | | | | | | | |
| Output Til Busy | | | C | | | | | | | | | | | | | |
| Generic IOCTL | | | | | | | | | | A | | | | | | B |
| Get Logical Dev | | | | | | | | | | B | | | | | | |
| Set Logical Dev | | | | | | | | | | B | | | | | | |
| IOCTL Query | | | | | | | | | A | | | | | | | |

**Dispositivos de: B:BLOQUE --- C:CARACTER --- A:AMBOS**

Tabla 3.3. Relación entre el campo de Atributo de un Controlador en DOS y los comandos que se invocan para el Controlador.

## Comunicación entre DOS y el Controlador.

**El flujo de control externo (es decir, aquel que se puede advertir observando trabajar al controlador de dispositivos como una caja negra) entre una aplicación DOS y un dispositivo periférico se muestra en la siguiente figura, resaltando el papel del controlador del dispositivo:**

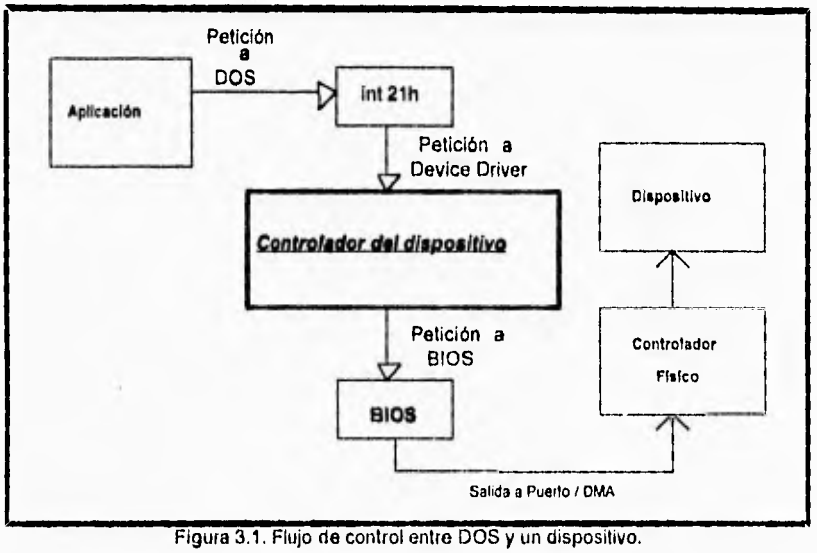



Figura 3.1. Flujo de control entre DOS y un dispositivo.

**Cuando la petición al dispositivo es generada a través de la interrupción 21h, DOS (o la aplicación desee hacer interfaz con el controlador) pasa al Controlador de dispositivos un paquete de datos, llamado Request Header (Encabezado de Petición, o Encabezado e Solicitud). El Controlador del dispositivo sabe donde está el Request Header porque DOS coloca la dirección de éste (en formato SEGMENTO:OFFSET) en los registros ES:BX.**

**El Request Header indica al controlador que acción (comando) debe realizar y la posición de los datos a leer o escribir (buffer de datos). El formato del Request Header es el mostrado en la tabla 3.4:**

| Campo | Longitud | Descripción |
|---|---|---|
| 1 | 1 | Longitud en bytes de este Request Header. |
| 2 | 1 | Código de unidad del dispositivo a manejar. |
| 3 | 1 | Código del comando a ejecutar. |
| 4 | 2 | Resultado de la operación (al terminar). |
| 5 | 8 | Reservado para DOS. |
| 6 | Variante | Depende del comando. |

Tabla 3.4. Formato del Encabezado de Solicitud.

- El primer campo indica la longitud total del Request Header.
- El segundo campo indica cual de los dispositivos que un Controlador maneja es el que se usará para ejercer el comando (Un Controlador de dispositivos de tipo bloque puede manejar más de un dispositivo, como en el caso del controlador de discos estándar en una microcomputadora).
- El tercer campo indica cual es el comando que se deberá realizar (más adelante en este mismo capítulo se describen los comandos estándar).
- El cuarto campo es llenado por el Controlador al terminar de procesar el comando para indicar a DOS si el resultado fue correcto o hubo algún error.
- El quinto campo está reservado por DOS, aunque la documentación técnica disponible no aclara si tiene un uso específico o solo se reserva para futuras extensiones a este estándar.
- El contenido y longitud del sexto campo varían dependiendo del comando que se esté ejecutando.

Cada comando enviado a un controlador envuelve en realidad dos pasos:

1. DOS llama a la rutina Estrategia.
2. DOS llama a la rutina Interrupción.

La llamada a Estrategia prepara el controlador para el comando que deberá realizar.
La rutina Interrupción, al ser llamada, usa la información dejada por la rutina Estrategia para realizar el trabajo del comando que le fue indicado a través del Request Header.
La razón básica para realizar esta división es distinguir entre una solicitud de transacción y el trabajo real de la transacción, lo cual en las versiones actuales de DOS (hasta la versión 6.21) no se utiliza para ningún fin práctico pero es una previsión de los diseñadores para extender a este Sistema operativo con futuras capacidades multitarea. La forma en que ésto se aplicaría para proveer la capacidad de multitarea es la siguiente:

Cuando una petición a algún controlador se recibe en la rutina Estrategia, ésta guarda la información acerca de la petición en una lista ligada que puede estar ordenada por prioridades, o por facilidad de acceso a dispositivo. De esta forma,

**cuando se realiza la llamada a la rutina Interrupción se atiende la petición que sea primera en la lista de peticiones al dispositivo, es decir aquella de mayor prioridad. Este esquema no ha sido aún implementado por DOS pero si por al menos otro Sistema operativo: OS/2 (Vease el capítulo dedicado a otros Sistemas operativos).**

**La única acción que tal vez toda rutina Estrategia realiza en DOS actualmente es la de salvar la dirección del Request Header (Guardada en el par de registros ES:BX) en variables locales dentro del Controlador de dispositivo, en donde puede ser accesada directamente por la rutina Interrupción. Ésto es a tal grado cierto que en algunas implementaciones del Sistema operativo DOS (como la escrita por IBM) todos los controladores de dispositivos estándares (los provistos por el fabricante en sus archivos de sistema para usarse con unidades de disco, monitor, teclado, puertos seriales y paralelos, etcétera) comparten una misma rutina Estrategia.**

**La figura 3.2 ilustra esta secuencia:**

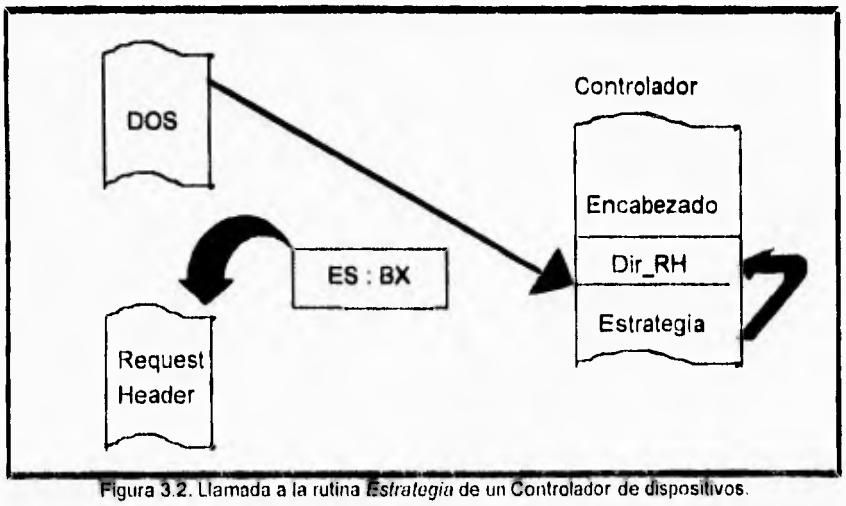



Figura 3.2. Llamada a la rutina *Estrategia* de un Controlador de dispositivos.

**A continuación DOS llama a la rutina *Interrupción*, la cual toma la dirección del *Request Header* (dejada en variables locales por la rutina *Estrategia*) e interpreta éste para averiguar el comando que se ejecutará y transfiere el control a la rutina de atención respectiva.**

**Los siguientes diagramas de flujo (Figura 3.3) muestran la secuencia en que DOS invoca a las rutinas *Estrategia* e *Interrupción*, así como la secuencia en que éstas operan internamente. De lado izquierdo se muestra el orden en que DOS realiza sus llamadas, mientras que en el extremo superior derecho se muestra la llamada**

**a la rutina *Estrategia*. Finalmente, en la parte inferior derecha se muestra la llamada a la rutina *Interrupción*.**

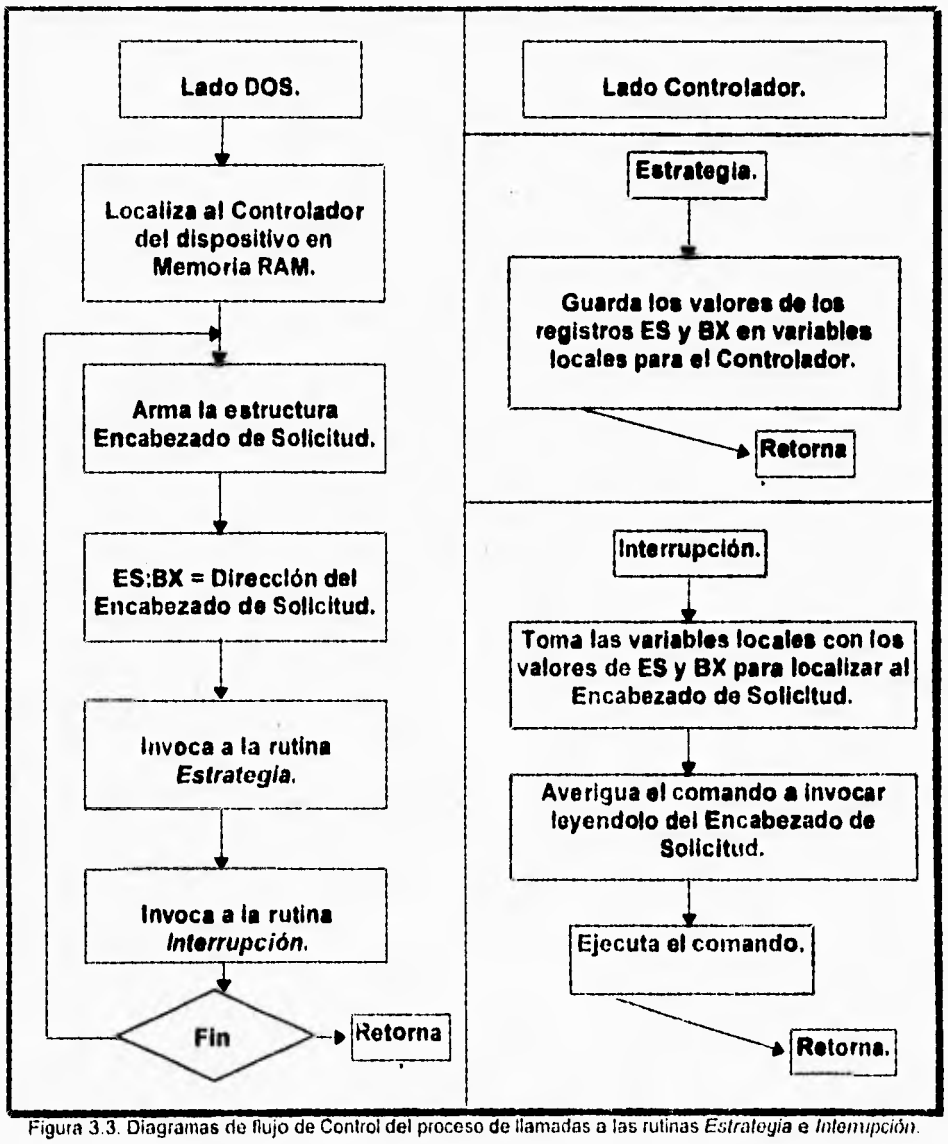



Figura 3.3. Diagramas de flujo de Control del proceso de llamadas a las rutinas *Estrategia* e *Interrupción*.

## Comandos enviados por DOS al Controlador.

**La siguiente tabla muestra los comandos enviados por DOS al Controlador definidos en la documentación técnica de DOS:**

| | | Soportado por | | |
|---|---|---|---|---|
| **Número** | **Nombre** | **Bloque** | **Car** | **Válido en la versión de DOS** |
| 0 | *Inicialization* | Si | Si | Todas |
| 1 | *Media Check* | Si | | Todas |
| 2 | *Get BIOS parameter Block* | Si | | Todas |
| 3 | *IOCTL Input* | Si | Si | Todas |
| 4 | *Input* | Si | Si | Todas |
| 5 | *Nondestructive Input* | | Si | Todas |
| 6 | *Input Status* | | Si | Todas |
| 7 | *Input Flush* | | Si | Todas |
| 8 | *Output* | Si | Si | Todas |
| 9 | *Output with Verify* | Si | Si | Todas |
| 10 | *Output Status* | | Si | Todas |
| 11 | *Output Flush* | | Si | Todas |
| 12 | *IOCTL Output* | Si | Si | Todas |
| 13 | *Device Open* | Si | Si | 3.0 y 3.10 |
| 14 | *Device Close* | Si | Si | 3.0 y 3.10 |
| 15 | *Removable Media* | Si | | 3.0 y 3.10 |
| 16 | *Output Til Busy* | | Si | 3.0 y 3.10 |
| 17 | No definido | | | 3.20 o mayor |
| 18 | No definido | | | 3.20 o mayor |
| 19 | *Generic IOCTL* | Si | | 3.20 o mayor |
| 20 | No definido | | | 3.20 o mayor |
| 21 | No definido | | | 3.20 o mayor |
| 22 | No definido | | | 3.20 o mayor |
| 23 | *Get Logical Device* | Si | Si | 3.20 o mayor |
| 24 | *Set Logical Device* | Si | Si | 3.20 o mayor |
| 25 | *IOCTL Query* | Si | Si | 5.0 o mayor |

Tabla 3.5. Comandos enviados por DOS al Controlador de dispositivos.

A continuación se da una breve explicación de cada comando:

### Comando 0 - *Initialization* (Inicialización).

El comando *Initialization* es siempre enviado por DOS al controlador inmediatamente después de que éste ha sido instalado en memoria. Ésto le permite al controlador realizar cualquier secuencia de acciones iniciales, como desplegar mensajes de aviso en pantalla o inicializar estructuras de datos. Un detalle de importancia es que, debido a que el controlador una vez instalado se considera parte de DOS, el comando *Inicialización* es el único desde el que es posible hacer llamadas a servicios de DOS debido a la naturaleza no Reentrante del Sistema operativo (DOS no puede llamar a DOS).

### Comando 1 - *Media Check* (Checa si el medio ha cambiado).

Este comando es exclusivo de los dispositivos en bloque, y es enviado por el Sistema Operativo DOS al Controlador para averiguar si el objeto ha cambiado. Un caso típico es el uso de unidades de disco flexibles, aunque no es privativo de éstas. Al retornar de la llamada DOS obtendrá uno de tres posibles valores, indicando que:

1. El objeto ha cambiado.
2. El objeto no ha cambiado.
3. El Controlador no sabe si el objeto ha cambiado o no.

### Comando 2 - *Get BPB* (Retorna bloque de parámetros de BIOS).

Este comando también es privativo de los dispositivos en bloque, y es llamado cuando el comando anterior retorna indicando que el objeto ha cambiado, o si el controlador no sabe si el medio ha cambiado. Su función es retornar al Sistema operativo un apuntador a información relevante del objeto controlado para ayudarle a localizar áreas criticas. Por ejemplo, en el caso de un disco esta información está contenida en el sector de arranque (boot sector) y contiene datos como el total de sectores por FAT, tamaño total del disco, etcétera.

### Comando 3 - *IOCTL Input* (Entrada de Control desde el dispositivo).

Este comando es enviado por el Sistema Operativo para obtener del Controlador información de control importante para la operación del Sistema. Por ejemplo, una aplicación podría solicitar a un controlador de impresión información acerca de la velocidad de los puertos de salida, estado actual de actividad, etcétera. Sin embargo, este comando tiene formato libre y por tanto la aplicación que lo emita y el controlador que lo reciba deben estar totalmente de acuerdo acerca de cual es la solicitud que se realiza.

### Comando 4 - *Input* (Entrada desde el dispositivo).

Este comando es emitido por el Sistema operetivo para solicitar al controlador una lectura del objeto controlado. Cuando el Sistema operativo recibe la información la turna a su vez a la aplicación que la haya solicitado.

### Comando 5 - *Nondestructive Input* (Revisa si hay entrada en el dispositivo).

Este comando se utiliza cuando una aplicación necesita saber si hay datos que leer, sin realmente leerlos.

### Comando 6 - *Input Status* (Averigua si se pueden leer datos).

El comando Input Status retorna información indicando si el dispositivo está listo para leer información o no. Siempre es llamado en los dispositivos de caracter antes del comando 4.

### Comando 7 - *Input Flush* (Limpia el buffer de entrada).

Esta comando permite a una aplicación descartar cualquiar entrada desde un dispositivo, limpiando el buffer de entrada del mismo.

### Comando 8 - *Output* (Salida hacia el dispositivo).

La función de este comando es inversa a la del comando 4, ya que se utiliza para escribir datos en un dispositivo. Estas dos son las llamadas más comúnes en cualquier dispositivo.

### Comando 9 - *Output with Verify* (Salida de control hacia el dispositivo).

El objetivo de este comando es el mismo que el del comando anterior. Sin embargo, se agrega comprobación de la información escrita antes de retornar. Su utilidad radica en el uso con dispositivos poco seguros.

### Comando 10 - *Output Status* (Averigua si se pueden escribir datos).

Este comando retorna al Sistema Operativo una bandera indicando si el dispositivo está o no listo para realizar escrituras. En dispositivos de caracter se llama siempre antes de realizar un comando 8.

### Comando 11 - *Output Flush* (Descarta información en buffer ).

Este comando provoca que el controlador descarte cualquier información que esté aún en espera de ser escrita en el dispositivo. Su utilidad es la cancelación rápida de ordenes de escrutura.

### Comando 12 - *IOCTL Output* (Salida de control hacia el dispositivo).

El comando 12 es usado para pasar al controlador información de control, en vez de datos. Mediante este comando se puede cambiar la forma en que el controlador se comporta. Por ejemplo, se puede indicar al controlador de puertos seriales que cambie la velocidad de entrada/salida, que utilice o deshabilite el puerto serial 2, etcétera.

### Comando 13 - *Device Open* (Abre dispositivo).

Este comando puede ser utilizado por el Controlador para llevar un control de la cantidad de veces que un dispositivo ha sido abierto. trabajando en conjunción con el comando 14, puede servir para prevenir ciertas condiciones que indiquen error, como el hecho de que los buffers se queden con información no escrita.

### Comando 14 - *Device Close* (Cierra dispositivo).

La función de este comando es la inversa a la del anterior, con el cual se utiliza para prevenir ciertas condiciones de error.

### Comando 15 - *Removable Media* (Verifica si el medio es removible).

Este comando pregunta al controlador si el objeto actual es removible. Su uso se restringe a los dispositivos de bloque.

### Comando 16 - *Output Until Busy* (Escribe hasta que el buffer se llene).

Este comando es utilizado en dispositivos de caracter solamente. Sirve para enviar datos a un dispositivo, pero solo hasta que el buffer de escritura se ha llenado. Un ejemplo tipico de ésto es el control de impresoras.

### Comando 19 - *Generic IOCTL* (Control de E/S estándar).

Este comando aparece a partir de la versión 3.2 de DOS, y su objetivo es proporcionar un estándar para realizar control de dispositivos. Su uso es poco común.

### Comando 23 - *Get Logical Device* (Obten la letra de un dispositivo).

Este comando fue agregado a partir de la versión de DOS 3.2, y tiene utilidad solo para los dispositivos de bloque. Tiene el propósito de indicar al Sistema Operativo si se utilizó alguna otra letra para accesar el dispositivo controlado (Capacidad que fue agregada en esta misma versión del Sistema operativo).

## Comando 24 - *Set Logical Device* (Asigna una letra a un dispositivo).

**Al igual que el anterior, este comando apareció desde la versión de DOS 3.2 y solo sirve para dispositivos de bloque. Se usa para designar otras letras mediante las cuales se pueda accesar un dispositivo determinado.**

## Comando 25 - *IOCTL Query* (Interroga acerca del control de E/S estándar).

**Este comando aparece a partir de la versión de DOS 5.0, y su objetivo es interrogar al controlador acerca de las funciones de IOCTL genérica (Vease el comando 19) que está preparado para soportar.**

**Para terminar este capítulo, se mostrará en un diagrama simplificado la estructura de un Controlador de dispositivos genérico:**

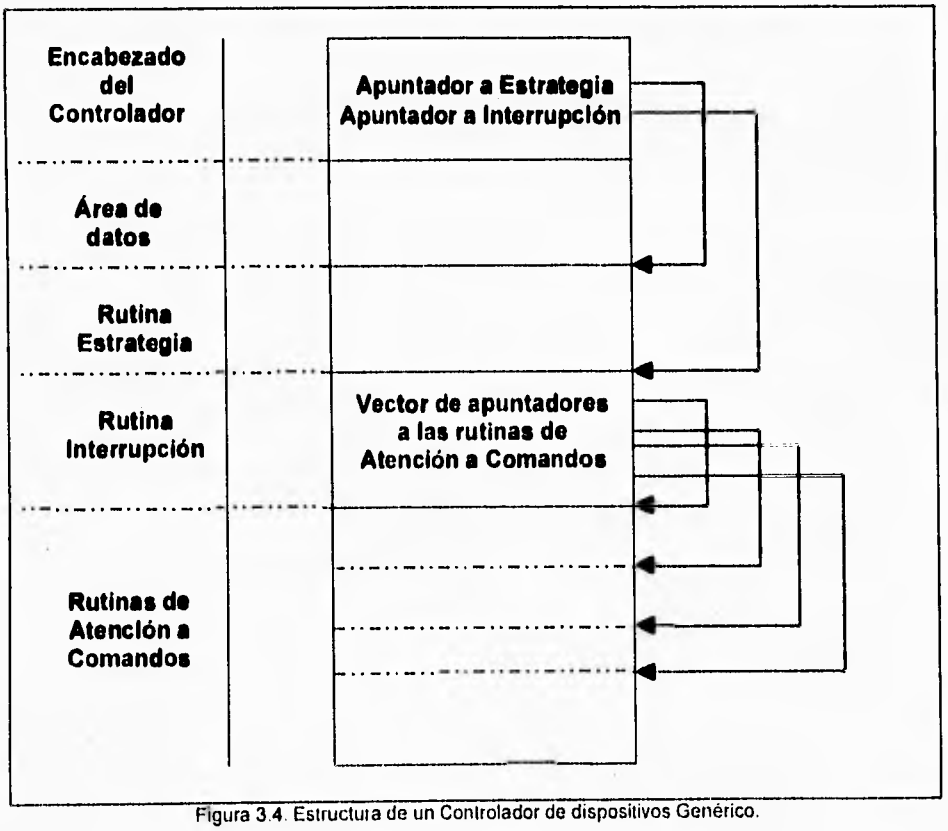



Figura 3.4. Estructura de un Controlador de dispositivos Genérico.

# 4. Proyecto: Controlador para un Disco Virtual bajo DOS.

Reporte del Análisis.

Reporte del Diseño.

Reporte de la Implementación.

## Lista de Tablas para el Capítulo 4.



## Lista de Figuras para el Capítulo 4.

# Capítulo 4

# Proyecto: Controlador para un Disco Virtual bajo DOS

## Reporte del Análisis.

### Antecedentes.

La Secretaría de Comercio y Fomento Industrial (SeCoFI) es una dependencia del Gobierno Federal que durante los últimos años se ha distinguido por ser uno de los organismos gubernamentales que ha dado el mayor impulso a la modernización informática de sus actividades. Este hecho implica la atención a factores como la capacitación a empleados, adquisición de equipo adecuado, actualización de paquetes de software adquiridos e impulso al desarrollo de productos propios, Modernización de la infraestructura de telecomunicaciones, etcétera.
El lector puede haber ya intuído que el proceso de cambio antes descrito en un ente tan grande implica problemas de diversa naturaleza, como por ejemplo la necesidad de implantar mecanismos de control y administración que puedan encauzar todo este crecimiento de manera concertada.
Respecto a esto último, una de las áreas en que se requieren métodos de control y administración más estrictos es la del control del Inventario de Software de la Secretaría.

### Definición del problema.

Antes de describir el entorno del problema, se debe hacer notar que el presente trabajo no tiene como objetivo presentar una alternativa de manejo de la situación que se describirá. Su alcance es mucho más modesto, pues pretende solo facilitar el actual método utilizado, aprovechando la infraestructura que se posee y presentando una aplicación de cierto interés y que a pesar de su tamaño represente un buen desafío técnico, computacionalmente hablando.

La situación es la siguiente:
Uno de los planteamientos base para la modernización informática de SeCoFI es la de la estandarización de su software. De este modo se definen las aplicaciones comerciales que serán las que se utilicen en toda la Secretaría, por supuesto con la flexibilidad suficiente para considerar de manera especial todos aquellos casos que así lo requieran. Por otro lado, se realizan desarrollos de productos de

software que por sus características faciliten funciones que sean comúnes a todas las áreas y se definen normas para su desarrollo.
Una vez hecho ésto, los originales de las aplicaciones comerciales adquiridas y copias de los productos desarrollados internamente se proporcionan a un Área de SeCoFI (la de Planeación e Informática), la cual se encarga de resguardar dichas aplicaciones y de proporcionarlas a otras instancias cuando así se requiera. Los casos en que un producto puede salir del almacén de Planeación e Informática son: 1) Préstamo y 2) Asignación del producto a un área de SeCoFI. De tal forma, otra manera en que un producto puede entrar al almacén es por medio de una Devolución. Éstas son las formas en que un producto de Software puede entrar o salir del Inventario. El Siguiente diagrama muestra esta situación:

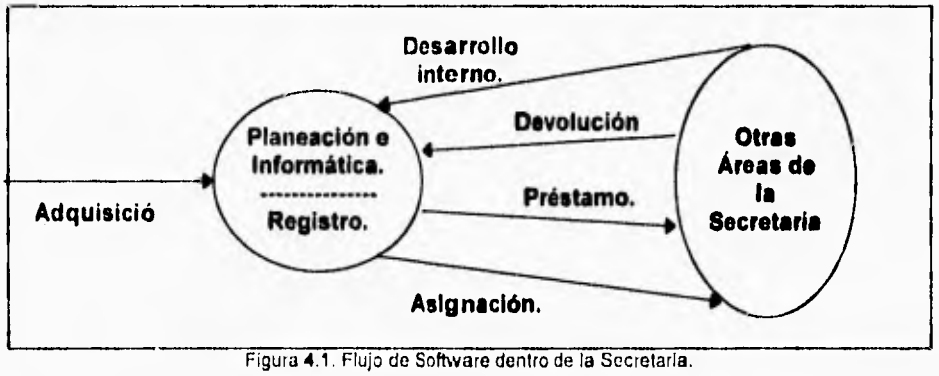



Figura 4.1. Flujo de Software dentro de la Secretaría.

Es así como existen sistemas de Registro y Control de Inventario que, desde un punto de vista administrativo, se encargan del registro de toda la información correspondiente a estos paquetes. Se está así en disponibilidad de conocer la situación actual del inventario, los nombres de destinatarios de un préstamo, las últimas versiones de un producto, registrar nuevas adquisiciones, etcétera.
Este esquema ha probado ser bastante eficiente durante el tiempo que ha operado (Aproximadamente dos años). Sin embargo, se han podido detectar algunas deficiencias durante este tiempo. A continuación se describirán aquellas que se considera motivan el desarrollo de la aplicación mostrada en el presente trabajo.
El Sistema está escrito en FoxPro en su versión para Red y actualmente se encuentra operando en una computadora PC bajo DOS en modo independiente (Esto implica que para efectos del Sistema de Registro, no se encuentra conectada a ninguna Red de computadoras), a pesar de que el ambiente en que se desarrolló facilitaría ésto.
Para realizar alguna de las transacciones de Entrada o Salida de Almacén que se plantearon en el Diagrama anterior se requiere que Personal de la Secretaría se traslade físicamente transportando consigo el material Informático desde o hasta el sitio en que se encuentra el Sistema de Registro (Está en el edificio de SeCoFI

en Periférico Sur), lo cual arriesga la integridad de la información y provoca pérdidas de tiempo y gastos en medios magnéticos (Discos o cintas).
Parte de estas dificultades se podría eliminar aprovechando la infraestructura de Telecomunicaciones que posee la Secretaría, la cual para la Ciudad de México está basada en antenas de transmisión Microondas y líneas telefónicas privadas, todo ésto enlazando aproximadamente 50 Servidores de Red local que ejecutan diversas versiones del Sistema operativo de Red Netware de Novell.
La Infraestructura de Comunicaciones de SeCoFI se muestra de manera simplificada en la siguiente imágen:

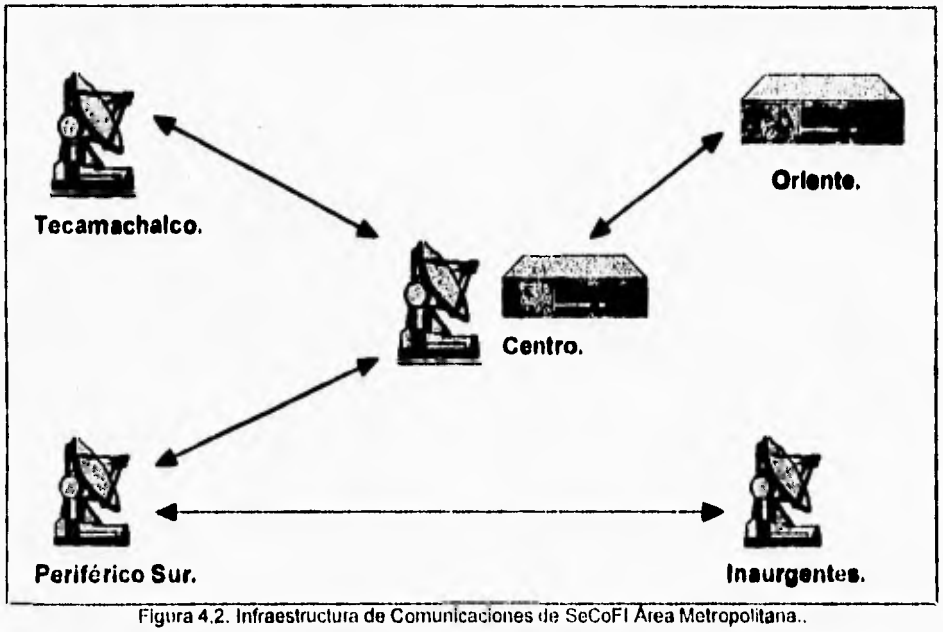



Figura 4.2. Infraestructura de Comunicaciones de SeCoFI Área Metropolitana..

Como ya se comentó, en ocasiones se padece de pérdidas en el inventario debido a múltiples causas (Como robo, borrado accidental, degradación de medios magnéticos, siniestros, etcétera), haciendose necesario el mantener copias excesivas de cada aplicación. De tal forma, se considera que es necesario contar con un nuevo método de control de dicha información (O modificar el ya existente) que reúna los siguientes requisitos:

1. Fácil acceso físico a la información.
Debe ser posible aplicar el método aún en condiciones críticas (Como errores en la comunicación de red, fallas en nodos intermedios, siniestros, etcétera). Para ésto, es recomendable que la plataforma en que se encuentre la solución sea sencilla de usar y administrar.

2. Facilidad y Transparencia en el manejo.
El método elegido, cualquiera que éste sea, debe ser muy fácil de usar. Es además determinante que los procesos de respaldo y recuperación de información puedan ser realizados de una manera lo más rápida y transparente posible para el usuario pues es una experiencia previa que en este medio todo lo que no cumple con dichos requisitos cae en el desuso.

3. Flexibilidad en la Creación, Recuperación y Actualización de información.
El método debe permitir accesar archivos de forma fácil y flexible. Debe también proporcionar la opción de recuperar el total o solo parte de los archivos de un paquete, de manera lógica y ordenada. En otras palabras, debe poderse obtener un solo archivo o todo un disco a través del empleo de dicho método. También debe permitir organizar jerárquicamente la información de tal forma que permita localizar rápidamente cualquier archivo.

4. Rapidez, Eficiencia, Economía.
El método debe permitir la administración y el acceso a la información de manera no secuencial, sino aleatoria. El método debe también ser rápido y ocupar poco espacio en memoria RAM cuando esté trabajando, además de tener pocas limitaciones en el tamaño y tipo de información que accese.

A continuación se representará de manera simplificada el Sistema de Inventario que actualmente se está utilizando:

## Descripción de la solución propuesta.

El problema que se ha descrito requiere para su solución la administración de recursos tanto humanos como materiales y la aplicación de una metodología de trabajo a largo plazo. Como ya se comentó al inicio de esta sección, La aplicación que se presenta en este trabajo no pretende resolver todo ese problema. Su alcance es mucho más modesto, pues solo pretende facilitar los procesos de manejo de la información que se maneja, información que por sus características está conformada por gran cantidad de archivos que están en un proceso de creación, actualización y desaparición permanente.

Se pretende que la aplicación desarrollada se integre al esquema de Control de Inventario de Software actual, al cual se le deberían realizar modificaciones para trabajo en Red y que se logre así utilizar de manera más provechosa la Infraestructura de Telecomunicaciones de la Secretaría.

A continuación se representará en un diagrama intencionalmente simplificado al máximo posible la estructura del Sistema de Registro actual, y después se presentará otra ilustración semejante que agregue los bloques correspondientes

**a nuestra aplicación. El propósito de ésto es permitir que se ubique de manera clara el contexto en que nuestra aplicación se ubica dentro del problema.**

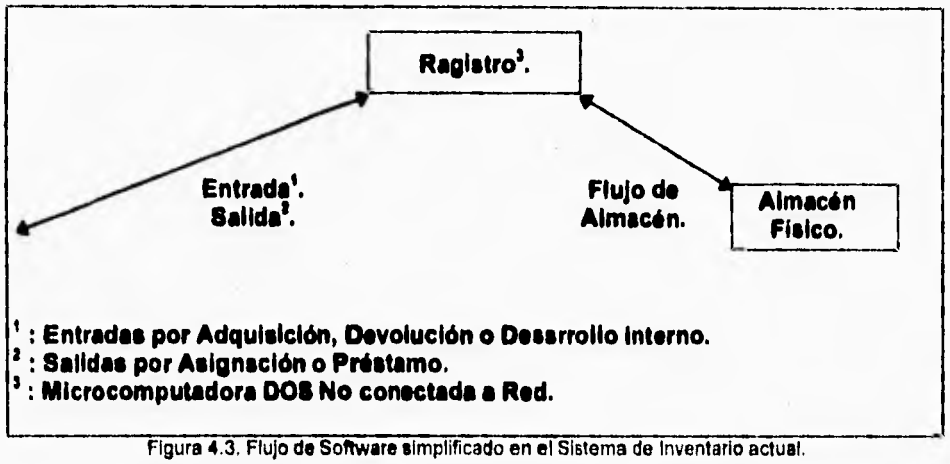



Figura 4.3. Flujo de Software simplificado en el Sistema de Inventario actual.

**El diagrama simplificado del Sistema de Inventario propuesto se muestra a continuación:**

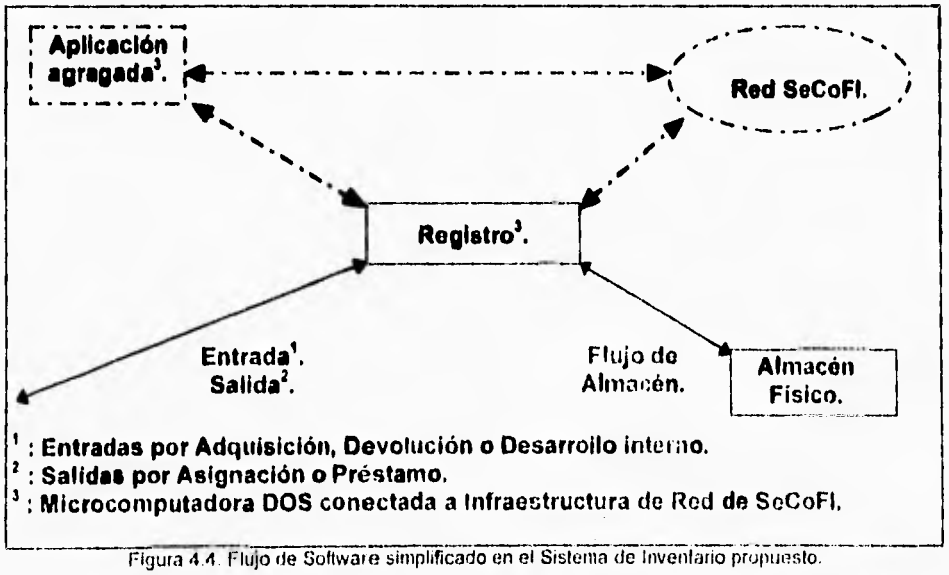



Figura 4.4. Flujo de Software simplificado en el Sistema de Inventario propuesto.

**Como se observa por la comparación entre los dos diagramas anteriores, se propone conservar el actual sistema realizando modificaciones para permitirle hacer uso de la Red SeCoFI y en particular de nuestra aplicación, la cual se encuentra corriendo en una microcomputadora PC unida a la Red, y que por lo tanto puede ser también accesada desde cualquier otro punto de ésta.**

**El acervo de Software concentrado en el almacén de SeCoFI está integrado por aplicaciones en disco flexible, manuales impresos y artículos como cables, tarjetas de interfaz, aplicaciones en memorias ROM, etcétera. Obviamente, lo más solicitado son aplicaciones en disco flexible, las cuales ocupan un espacio total de aproximadamente 1.5 Gigabytes. Sin embargo, la experiencia indica que un alto porcentaje de las Entradas y Salidas del almacén (Arriba del 90%) se restringe a un conjunto bien delimitado de aplicaciones (Abajo del 15% del total), mientras que el resto se requiere solo muy eventualmente. Es decir, la mayor cantidad del tráfico del inventario de almacén se limita a aproximadamente 235 Megabytes de datos. La siguiente ilustración indica estas relaciones:**

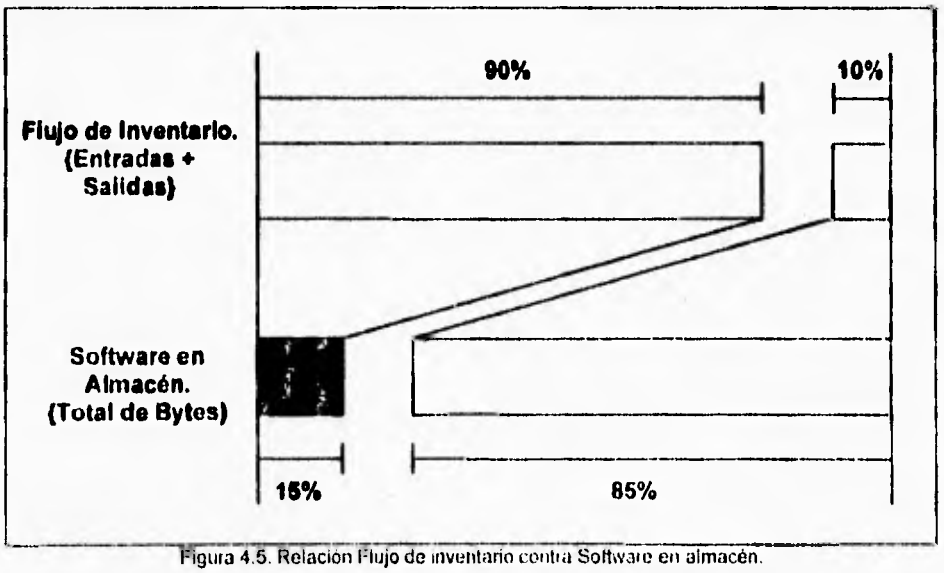



Figura 4.5. Relación Flujo de inventario contra Software en almacén.

Del diagrama anterior es evidente que si se agiliza el Flujo de inventario del 15% de aplicaciones indicado la operación del Sistema completo se eficientizará. El objetivo de nuestra aplicación es proporcionar acceso a ese 15% (Aproximadamente 235 Megabytes) cumpliendo con los 4 requisitos antes

**mencionados. Para ésto, lo más adecuado es destinar 235 Megabytes de un disco duro en la microcomputadora de nuestra aplicación para almacenar este 15% de código.**

**Dado el planteamiento anterior, la propuesta de solución es la siguiente:**

1. **Se propone la creación de un conjunto de Librerías de archivos donde cada una de éstas contenga una "imágen" del software de una o más de las aplicaciones que se encuentran en el 15% de alta frecuencia de Entrada/Salida de almacén.**
2. **Se plantea que dichas librerías residan en el disco duro de la estación de trabajo en que resida nuestra aplicación, de tal forma que sean de fácil acceso. Las librerías serían archivos binarios comúnes y corrientes, de tal forma que puedan residir en un sistema de archivos del sistema operativo MS-DOS como cualquier otro archivo.**
3. **El resto del esquema que se propone radica en la implementación de un sistema que simule un disco duro a partir de una librería de software determinada, de tal forma que sea posible accesar el contenido de dicha librería a través de una letra de unidad lógica cualquiera del Sistema Operativo de la misma forma en que se haría con un disco duro o con el disco flexible dentro de la unidad A: de la microcomputadora. Sin embargo, la librería sería solo un archivo para el resto del sistema operativo. Es decir, la librería sería un disco virtual.**

**La siguiente ilustración muestra la forma en que dicho esquema se implementaría:**

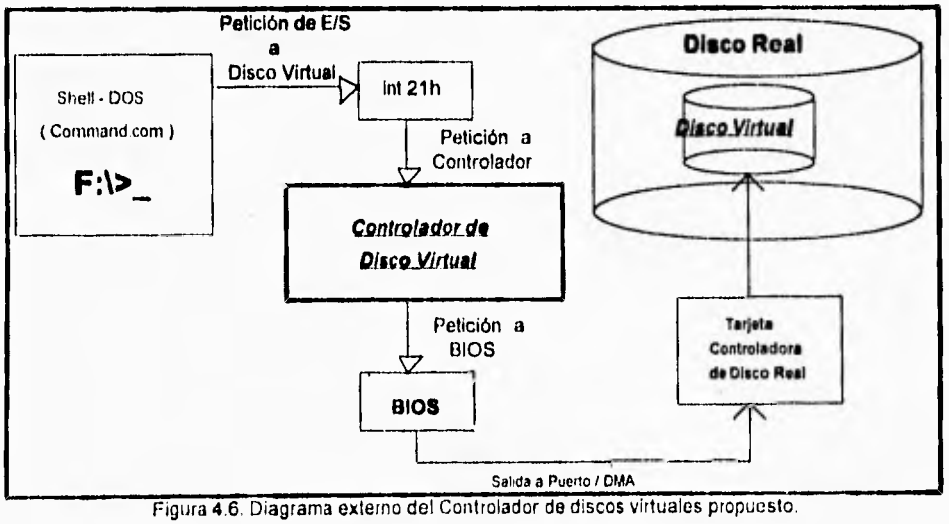



Figura 4.6. Diagrama externo del Controlador de discos virtuales propuesto.

**Como se advierte en la figura, la interfaz a las librerías (y parte central de la solución) es un Controlador de dispositivos (Device Driver) cuyo objetivo es permitir el acceso a los contenidos de una librería específica (conteniendo un conjunto de archivos que conforman una o más aplicaciones) por medio de una unidad lógica, la cual en la figura mostrada está representada por la letra F:. Éato es semejante al efecto de simular un disco por medio de memoria RAM (discos virtuales), con la diferencia que en este caso el medio de almacenamiento es no volátil.**

## Justificación de la solución propuesta.

**Analizando la solución propuesta se observa lo siguiente:**

1. **Para el requisito número 1 (Fácil acceso físico a la información): La aplicación estaría basada en cualquier microcomputadora con DOS de versión 3.0 o posterior instalado. Se considera que esta plataforma es de fácil acceso.**
2. **Para el requisito número 2 (Facilidad y Transparencia en el manejo): La facilidad y transparencia de manejo se cumplen dado que cualquier usuario acostumbrado a trabajar con el sistema operativo DOS sería capaz de accesar la librería por medio de la interfaz descrita.**
3. **Para el requisito número 3 (Flexibilidad en la Creación, Recuperación y Actualización de información): Se agregaría una utilería independiente del Controlador propuesto que se utilizaría para crear librerías de un tamaño determinado, y que fuera de fácil uso. Por otro lado, dado que un archivo de librería podría verse y utilizarse como si fuera el directorio principal de un Sistema de archivos DOS cualquiera, se podría accesar cualquier archivo dentro de la librería utilizando los comandos DOS para manejo de archivos y directorios convencionales, o incluso utilerías para manejo de disco como PC-TOOLS u otras.**
4. **Para el requisito número 4 (Rapidez, Eficiencia, Economía): El método de acceso usado por el Controlador de disco virtual es inherentemente aleatorio, lo cual lo hace más veloz. Por otro lado, el código estaría escrito en lenguajes C y Ensamblador, lo cual favorece su rapidez de ejecución. Además, este último hecho favorece el tamaño final del Controlador (La mayor parte del cual debe residir en memoria RAM mientras el equipo permanezca trabajando). Se calcula, por las pruebas realizadas, que el tamaño final del Controlador de disco virtual será de aproximadamente 10 KiloBytes.**

## Restricciones de la solución propuesta.

**Como cualquier otro, el método propuesto tiene ciertas limitantes. Las más importantes son:**

1. **El controlador implementado ocupa memoria base una vez que ha sido instalado. Se ocuparían aproximadamente 10 KiloBytes de memoria RAM (Las**

últimas versiones del Sistema operativo DOS permiten subir Controladores de dispositivos a memoria alta, liberando así memoria base del equipo).
2. Una vez conteniendo información, las librerías no pueden cambiar su tamaño, por tanto deben estar siempre tan llenas de información como sea posible a fin de evitar el desperdicio de espacio útil.
3. Una de las restricciones en la implementación del controlador es que se deben realizar accesos a disco a nivel BIOS, lo cual implica no poder usar las facilidades de acceso a disco proporcionadas por DOS. Ésto implica que la geometría del disco debe ser interpretada de manera totalmente manual, lo cual incrementa el nivel de dificultad del desarrollo.
4. La restricción descrita en el punto anterior implica también que una librería no puede ser accesada dentro de otro dispositivo de almacenamiento físico que no sea un disco duro (como por ejemplo unidades de red, dispositivos de cinta, etcétera). Para lograr ésto sería necesario escribir un controlador con las capacidades específicas para hacer ésto*.
5. En la unidad lógica simulada no se deben ejecutar comandos o aplicaciones que por su naturaleza no sean compatibles con la Filosofía de un dispositivo virtual. Por ejemplo, el programa FORMAT realiza accesos BIOS a disco y podría dañar a los discos virtual y real si se ejecuta sobre una unidad virtual. Esto puede compararse a tratar de leer datos del dispositivo de video, escribir datos al de manejo de teclado, u otras contradicciones semejantes.
6. El controlador resultante no será compatible con la tecnología de compresión de disco en tiempo real que apareció en la versión 6.0 de MS-DOS, puesto que éste utiliza una técnica semejante.

## Descripción externa del Sistema propuesto.

El objetivo de esta sección es definir el aspecto que el Controlador de discos virtuales propuesto deberá presentar al usuario. En otras palabras, se tratará de definir la interfaz a usuario. La forma en que dicha interfaz sea implementada se describirá en la sección de Diseño.

1. El sistema resultante será un archivo Controlador de Dispositivos que se apegará a la estructura definida y requerida por el Sistema operativo MS-DOS, y podrá instalarse y ejecutarse consistentemente sobre sus versiones 3.0 y posteriores, o aquellas de otros proveedores de Sistemas operativos DOS que conformen a este estándar de manera completa.
2. Por simplicidad, los archivos (o librerías) que se emplearán como discos virtuales deberán estar siempre en el directorio raíz del primer disco duro de la

* A este respecto se puede ejemplificar con programas como el *Mscdex.com* para el acceso a una unidad de disco compacto o *Netx.com* en Novell, el cual proporciona al usuario la facilidad de ver áreas del disco servidor como unidades de disco lógicas. Este último, sin embargo sigue una filosofía distinta a la de un Controlador de dispositivos.

computadora en que el controlador se instale. Es importante hacer notar que estas limitaciones pueden superarse en un diseño futuro.

3. El Controlador de discos virtuales podrá soportar y manejar un máximo de 5 unidades lógicas simuladas, aunque pueda trabajar con solo una. Estas capacidades serán configuradas a través de la línea de instalación del Controlador dentro del archivo Config.Sys.
4. La forma de indicar al controlador cuántas unidades soportará por instalación y qué archivos tomará para la simulación será a través de la línea de instalación del controlador, en el archivo config.sys.
5. Los parámetros que la línea de instalación aceptará son:
   /d: Indica el máximo de discos que el controlador simulará.
   /a: Indica un archivo que simulará un disco.
   - Si el parámetro /d: es seguido por un entero que exceda de 5, el valor se truncará a 5. Si, por otro lado, es cero o se omite entonces se tomará un valor que iguale a la cantidad de archivos declarados usando su parámetro /a. No se aceptan valores negativos.
   - El parámetro /a: debe ser seguido por un nombre de archivo que cumpla con las restricciones de estar en la raiz del primer disco de la microcomputadora en que se trabaje.
   - Si la cantidad de archivos declarados excede a la de discos declarados se tomarán archivos hasta que se complete la cantidad de discos señalada y el resto se desechará.
   - Si alguno de los archivos declarados no existe la instalación será cancelada.
   - Si la cantidad de Discos declarados excede a la de archivos declarados se instalarán solo tantas unidades lógicas como archivos se hayan declarado.

Por ejemplo, en el archivo config.sys de la computadora podria tenerse la línea:

**Device = c:\tesis\codigo\simula.sys /a:win.dsc /d:7 /a:exc.dsc**

La cual instalaria el controlador creando 2 unidades y asignando el archivo win.dsc al primero y exc.dsc al segundo. Asi, si la computadora tiene un disco duro con dos particiones el controlador definiría las unidades E: y F:. El archivo win.dsc seria accesado a través de la unidad E: y el exc.dsc a través de la F:.

6. Se desarrollará otra aplicación que automatice el proceso de creación de los archivos que sirvan como discos virtuales, y cuya interfaz permita definir el tamaño que el disco definido tenga. Su formato debe ser el siguiente:

   **creadisk <Nombre> <Tamaño>**

   donde <Nombre> indica el nombre del archivo que se creará y <Tamaño> indica el tamaño en MegaBytes.
7. Se deberá proveer además una aplicación que permita crear una imágen a partir de un disquete, de manera que dicha imágen pueda ser accesada a partir de una unidad definida por el usuario del controlador.

computadora en que el controlador se instale. Es importante hacer notar que estas limitaciones pueden superarse en un diseño futuro.

3. El Controlador de discos virtuales podrá soportar y manejar un máximo de 5 unidades lógicas simuladas, aunque pueda trabajar con solo una. Estas capacidades serán configuradas a través de la línea de instalación del Controlador dentro del archivo Config.Sys.
4. La forma de indicar al controlador cuántas unidades soportará por instalación y qué archivos tomará para la simulación será a través de la línea de instalación del controlador, en el archivo config.sys.
5. Los parámetros que la línea de instalación aceptará son:
   /d: Indica el máximo de discos que el controlador simulará.
   /a: Indica un archivo que simulará un disco.
   - Si el parámetro /d: es seguido por un entero que exceda de 5, el valor se truncará a 5. Si, por otro lado, es cero o se omite entonces se tomará un valor que iguale a la cantidad de archivos declarados usando su parámetro /a. No se aceptan valores negativos.
   - El parámetro /a: debe ser seguido por un nombre de archivo que cumpla con las restricciones de estar en la raíz del primer disco de la microcomputadora en que se trabaje.
   - Si la cantidad de archivos declarados excede a la de discos declarados se tomarán archivos hasta que se complete la cantidad de discos señalada y el resto se desechará.
   - Si alguno de los archivos declarados no existe la instalación será cancelada.
   - Si la cantidad de Discos declarados excede a la de archivos declarados se instalarán solo tantas unidades lógicas como archivos se hayan declarado.

Por ejemplo, en el archivo config.sys de la computadora podría tenerse la línea:

**Device = c:\tesis\codigo\simula.sys /a:win.dsc /d:7 /a:exc.dsc**

La cual instalaría el controlador creando 2 unidades y asignando el archivo win.dsc al primero y exc.dsc al segundo. Así, si la computadora tiene un disco duro con dos particiones el controlador definiría las unidades E: y F:. El archivo win.dsc sería accesado a través de la unidad E: y el exc.dsc a través de la F:.

6. Se desarrollará otra aplicación que automatice el proceso de creación de los archivos que sirvan como discos virtuales, y cuya interfaz permita definir el tamaño que el disco definido tenga. Su formato debe ser el siguiente:

   **creadisk <Nombre> <Tamaño>**

   donde <Nombre> indica el nombre del archivo que se creará y <Tamaño> indica el tamaño en MegaBytes.
7. Se deberá proveer además una aplicación que permita crear una imágen a partir de un disquete, de manera que dicha imágen pueda ser accesada a partir de una unidad definida por el usuario del controlador.

8. Para terminar, cualquier unidad definida y usada por el controlador se deberá comportar como una unidad estándar en el sistema.

## Otros casos semejantes.

Se pueden encontrar ejemplos de planteamientos semejantes en la literatura de la computación. Por brevedad se citarán solo tres de éstos.

- El sistema operativo VM (Virtual Machine) de IBM fue creado bajo el concepto de máquinas virtuales. Es capaz de alojar más de un sistema operativo (Tal es el caso de AIX, el UNIX de IBM), dando la ilusión de que los usuarios de cada uno están trabajando en un ambiente totalmente independiente. VM es capaz de segmentar un solo disco duro en varios "discos virtuales" que no son más que una secuencia de sectores asignados a una unidad lógica.

- Bajo el sistema operativo UNIX versión V se han hecho previsiones para facilitar la comunicación de información con otros sistemas operativos de uso común, tales como Macintosh y MS-DOS. Con respecto a este último, es posible asignar un disco estilo DOS a una zona de disco que se encuentre bajo un sistema de archivos UNIX. Dicho de otra forma, se puede "montar" un Sistema de archivos DOS en un Sistema de archivos UNIX. Una vez hecho ésto, todos los comandos UNIX serán válidos para el disco DOS. La sintáxis para lograr ésto con el diaquete en la unidad A: de un sistema UNIX es la siguiente:

  **# mount -F dos /dev/dsk/f0t /mnt**

  Si se quiere realizar el mismo procedimiento con la partición DOS del primer disco duro del sistema basta con utilizar /dev/dsk/0s5 como el nombre del dispositivo a usar.

- A partir de la versión 6.0 del sistema operativo MS-DOS es posible comprimir información. Este efecto se logra al aumentar un controlador de dispositivos que asignará una unidad extra a un archivo residente en el disco duro de la computadora. Si el usuario ve el archivo a través de la unidad extra supondrá que tiene un disco de mayor capacidad de la real. Este principio es básicamente el mismo que se aplica en el presente trabajo.

- En esta misma versión de DOS se provee una aplicación llamada *InterLink*,cuyo objetivo es la compartición de recursos de memoria secundaria y de impresión entre dos o más computadoras conectadas entre si mediante sus interfaces paralela o serial. Por ejemplo, se puede accesar la unidad de disco flexible de una computadora desde otra mediante un cable que conecte sus puertos seriales. La computadora Servidor (la que proporciona el recurso a compartir) trabaja con un programa que la obliga a estar en modo dedicado. En las computadoras Cliente se instala un Controlador de Dispositivos que les permite utilizar de manera transparente los recursos del Servidor

accesandolos mediante nombres de unidad de bloque convencionales, como E:, R:, AUX:, etcétera.

## Análisis Arquitectónico.

El resto de la Sección dedicada al Reporte de Análisis está centrado en la Descripción estructural del Controlador desarrollado. Para cumplir dicho objetivo, se decidió utilizar Técnicas simplificadas de Análisis orientado a Objetos, las cuales se considera que poseen, entre otros, los siguientes atributos:

- Definen con claridad cada una de las entidades (objetos) que intervienen en un problema, así como sus atributos.
- Establecen las Relaciones que se dan entre dichas entidades, definiendo en conjunto la estructura estática (Modelo de Información) del Sistema.
- Describen el comportamiento dinámico de cada una de las entidades que intervienen en el problema (Por medio de diagramas de estados), facilitando así el estudio de Sistemas en Tiempo real.
- Permiten definir con claridad las acciones que cada entidad deberá realizar bajo determinadas condiciones, simplificando así el paso a la fase de diseño.

Probablemente el atributo de esta Metodología que más favorece a nuestro proyecto es la libertad que proporciona al desarrollador para elegir los métodos de diseño e implementación que desee, sin forzosamente conservar una metodología orientada a objetos. La importancia de este atributo es que el uso de una técnica de implementación orientada a objetos resulta prohibitiva en nuestro caso debido a los requerimientos de estructura que DOS impone a un Controlador de dispositivos. En otras palabras, habría sido más difícil resolver los conflictos de direccionamiento de memoria que surgieron.
La Nomenclatura utilizada para representar el Modelo de Información es:

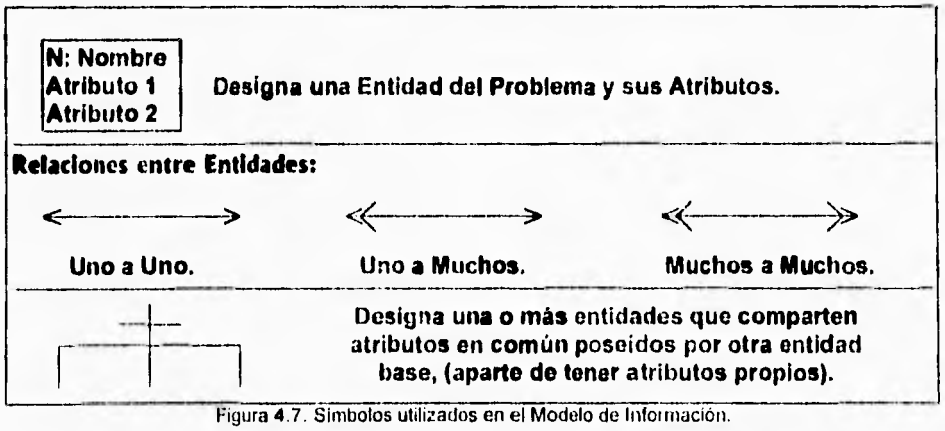



Figura 4.7. Símbolos utilizados en el Modelo de Información.

## Modelo de Información.

**El Modelo de Información que representa a las entidades relacionadas con la construcción u operación del Controlador es el siguiente:**

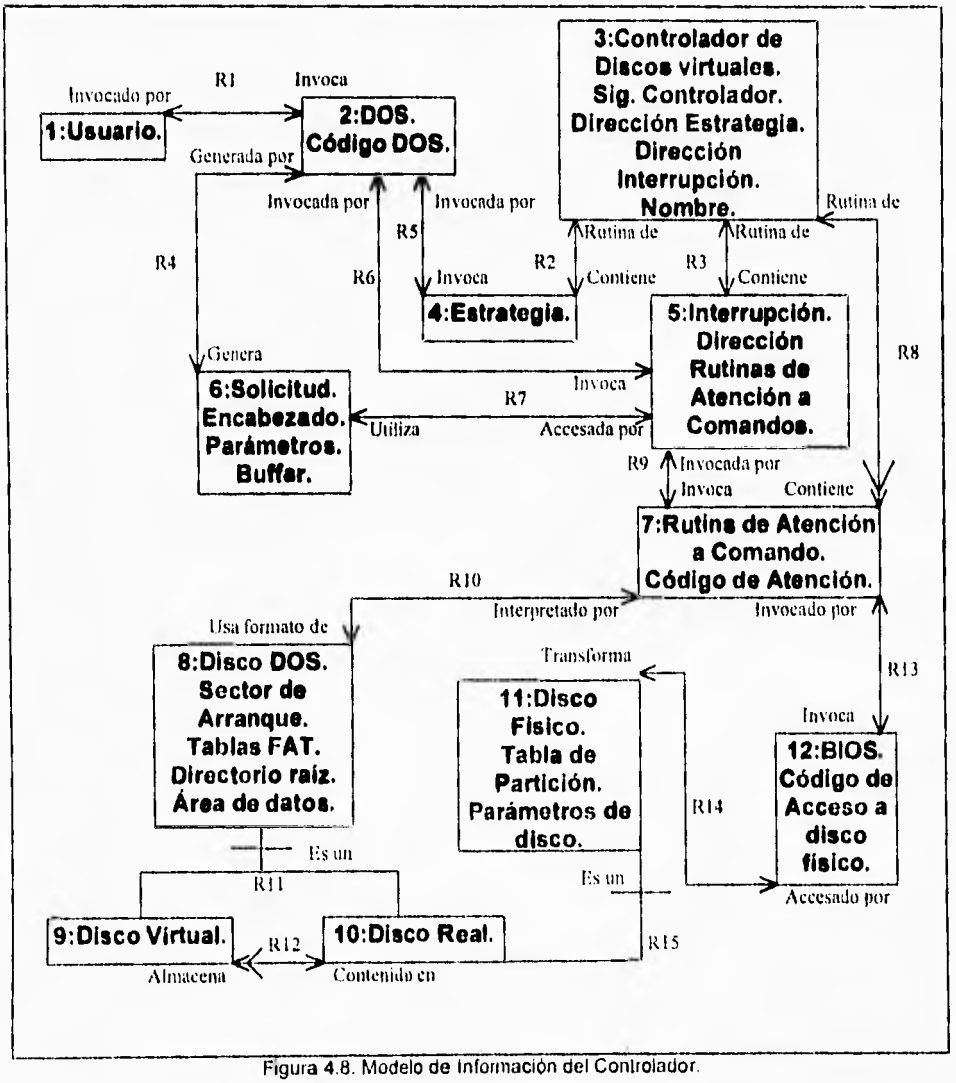



Figura 4.8. Modelo de Información del Controlador.

A continuación se define y explica cada una de las entidades y relaciones representadas.

## Definición de Entidades.

1: Usuario.
Representa a un solicitante de servicio relacionado con un Disco virtual. Se considera como una Entidad externa a nuestro y se incluye solo para facilitar la comprensión de éste.

2: DOS.
Conformado por el Sistema operativo y sus Interfaces para programación de aplicaciones, en lo que respecta a nuestro problema se circunscribe por un lado a las rutinas (invocadas por medio de interrupciones de DOS) que proveen al Usuario con acceso a las Unidades virtuales y por otro al manejo del Controlador que detalla su estructura.

3: Controlador de Discos virtuales.
Es el programa desarrollado en este trabajo. Provee a DOS con una interfaz estándar para el acceso a nuestros Discos virtuales. Contiene apuntadores al Siguiente Controlador instalado en Memoria y a las rutinas Estrategia e Interrupción, Además de una Cadena que indica su nombre o la cantidad de dispositivos que soporta.

4: Estrategia.
Rutina que es parte del código del Controlador y cuya existencia y comportamiento están definidos por la Definición de Interfaz estándar a Controladores de Dispositivos en DOS.

5: Interrupción.
Rutina existente en todo Controlador de Dispositivos, cuyo objetivo principal es invocar a la Rutina de Atención para el Comando en turno. Contiene apuntadores a todas las rutinas de Atención a Comandos que el Controlador soporta.

6: Solicitud.
Estructura de datos generada por DOS cada vez que el Usuario requiere un Servicio relacionado con un Disco virtual. Está conformada por variables utilizadas para pasar parámetros a las rutinas del Controlador y para conocer resultados.

7: Rutina de Atención a Comando.
Cada Rutina de Atención a Comando implementa directamente parte del manejo del Disco virtual.

8: Disco DOS.
Es una Entidad abstracta cuya finalidad es describir la estructura que posee un Sistema de archivos DOS. Tanto el Disco real como los Discos virtuales poseen dicha estructura.

**9:** **Disco Virtual.**
**Es la entidad accesada por nuestro Controlador. Fisicamente está conformado por un Archivo binario que reside en el Sistema de archivos DOS del Disco real.**

**10:** **Disco Real.**
**Es el disco que tiene formato DOS y que se utiliza para almacenar los archivos que representan Discos virtuales.**

**11:** **Disco Fisico.**
**Es el medio de almacenamiento de bajo nivel utilizado por un disco DOS. Se caracteriza por un Sector de Partición (Independiente del Sistema operativo) y por 3 parámetros que describen su Geometría y capacidad (Dichos parámetros son: Número de cabezas de Lectura/Escritura, Número de Cilindros y Número de sectores por pista).**

**12:** **BIOS.**
**Es el conjunto de rutinas registradas en Memoria de Solo lectura que definen una interfaz estándar (En realidad ésto depende ligeramente del implementador de cada BIOS) para la realización de actividades de Entrada/Salida básicas. En lo que atañe a nuestro sistema, las rutinas de más uso son aquellas de manejo de Disco físico y las de Salida a terminal de video.**

## Definición de Relaciones.

**R1.**
**Indica la Invocación que realiza un Usuario a DOS para acceder a un Disco virtual, y que se realiza a través de una interrupción.**
**R2.**
**El código de la rutina Estrategia está dentro del Controlador.**
**R3.**
**El código de la rutina Interrupción está dentro del Controlador.**
**R4.**
**DOS genera una Estructura de datos llamada Solicitud cada vez que se le indica realizar una operación sobre un Disco virtual.**
**R5.**
**DOS invoca a la rutina Estrategia para permitirle almacenar la Dirección del Encabezado de Solicitud en una variable del área de datos del Controlador.**
**R6.**
**DOS invoca a la rutina Interrupción, la cual a su vez localiza a la Solicitud generada por DOS mediante la variable del Controlador instanciada por la rutina Estrategia, y que contiene la dirección de la Solicitud.**
**R7.**
**La rutina Interrupción utiliza a la Solicitud creada por DOS para averiguar el comando que se solicita.**
**R8.**

**El controlador contiene el código de las rutinas de Atención a Comandos.**
**R9.**
**Una vez conociendo el comando que se debe ejecutar, la rutina Interrupción invoca a la Rutina de Atención a Comando correspondiente.**
**R10.**
**Las rutinas de Atención a Comando Hacen uso del formato de disco DOS de los Discos real y virtual para traducir direcciones lógicas a físicas, las cuales se utilizan para acceder al Disco físico.**
**R11.**
**Los Discos real y virtual cumplen ambos con la estructura propia de un Disco DOS. Es decir, poseen un sector de arranque, al menos una Tabla de partición, un directorio raiz y un área de datos.**
**R12.**
**Uno o más discos virtuales están contenidos en un Disco real.**
**R13.**
**Una vez que se han traducido las direcciones lógicas y se tienen las correspondientes físicas, las rutinas de Atención a Comandos pueden invocar a BIOS para acceder al Disco físico.**
**R14.**
**BIOS afecta directamente al disco físico, al realizar sobre éste operaciones de lectura, escritura, interrogación de estado, etcétera. BIOS es una Entidad externa que se incluye solo para clarificar al problema y ubicar el sitio en que se realiza su acceso a Disco físico.**
**R15.**
**Cada Disco real debe corresponder siempre a un Disco físico, el cual aloje un Sistema de archivos en formato DOS en por lo menos su primera partición.**

## Modelos de Estados.

**La utilización de Modelos de Información durante la fase de Análisis es útil para ilustrar las relaciones estructurales (Arquitectónicas) existentes entre los distintos elementos que componen un Sistema. Sin embargo, su objetivo se circunscribe a mostrar solo el caracter estático de éste. Normalmente las entidades definidas por el Diseñador del Sistema y representadas en un Modelo de Información poseen conductas dinámicas, es decir, variantes en función del tiempo. De tal forma, una entidad puede existir desde el comienzo de la ejecución de un Sistema o bien crearse y destruirse a intervalos de tiempo regulares o incluso aleatorios. Así mismo, puede permanecer permanentemente estática o responder constantemente a eventos generados por sucesos tales como una orden de la Interfaz a Usuario, cumplimiento de una serie de condiciones internas, interrupciones del hardware en que se ejecute el programa, etcétera. Para representar la Conducta dinámica de las entidades de un Sistema se utilizan Modelos de Estados, también conocidos en la Literatura de la Computación como Máquinas de Estados. Un ejemplo breve de Modelo de Estados es el siguiente:**

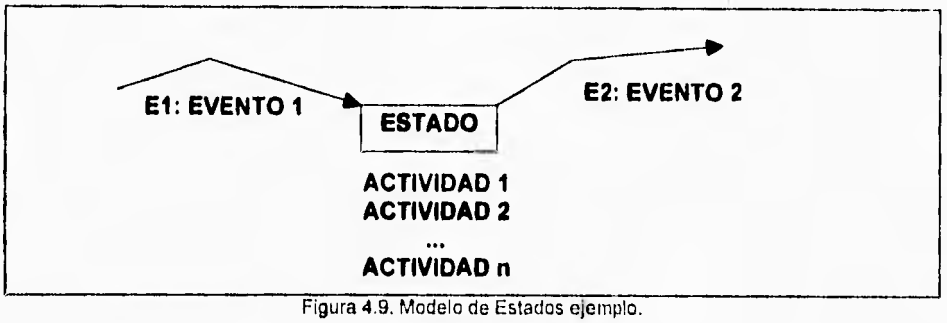



Figura 4.9. Modelo de Estados ejemplo.

**Dada la naturaleza del Sistema que se desarrolla en esta aplicación, se considera necesario describir al menos de manera superficial las características dinámicas de sus componentes más importantes.**

**El Modelo de Estados para la Entidad DOS es el siguiente:**

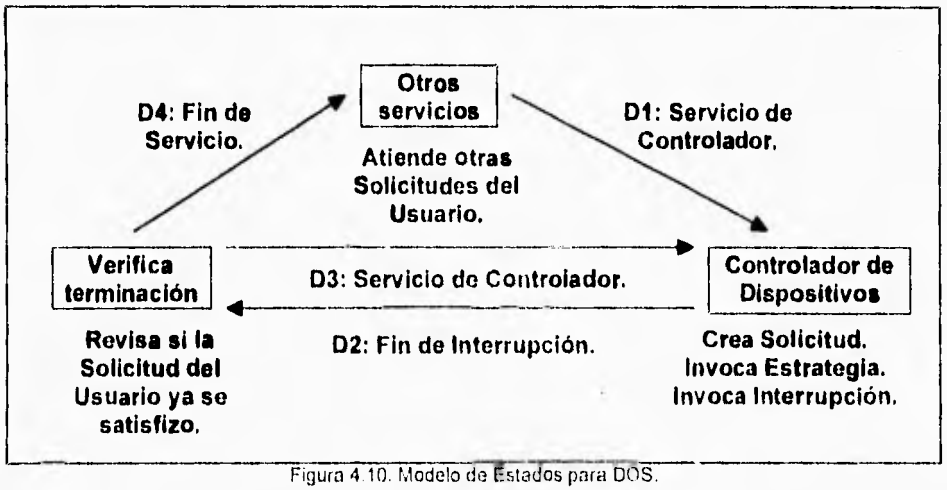



Figura 4.10. Modelo de Estados para DOS.

**Bajo condiciones normales, DOS constantemente se encuentra atendiendo Solicitudes de servicio por parte de programas usuarios. Cuando una de estas Solicitudes involucra a nuestro Controlador DOS crea una Solicitud e invoca sucesivamente a las rutinas Estrategia e Interrupción. Si la solictud se satisfizo con esta llamada DOS avisa el resultado al usuario. De lo contrario repite los pasos anteriores hasta terminar.**

**El Modelo de Estados para la Entidad Controlador de Discos virtuales es el siguiente:**

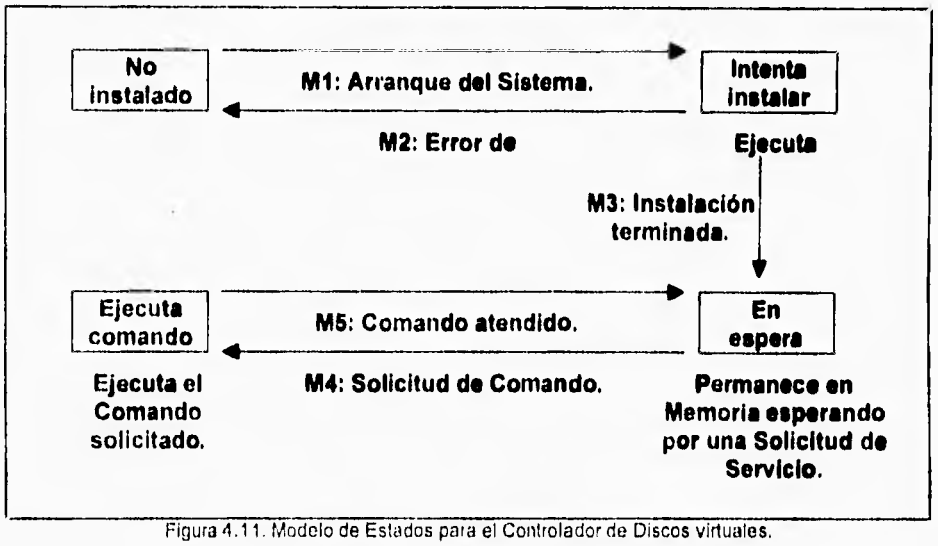



Figura 4.11. Modelo de Estados para el Controlador de Discos virtuales.

**Al arrancar la computadora DOS invoca el Comando 0 (Inicialización) del Controlador. Si este falla la instalación del Controlador se cancela. De lo contrario se considera instalado. A partir de ese momento estará en memoria esperando por una Solicitud de servicio que lo involucre. La atenderá y volverá a su estado de espera.**

**El Modelo de Estados para la Entidad Solicitud es el siguiente:**

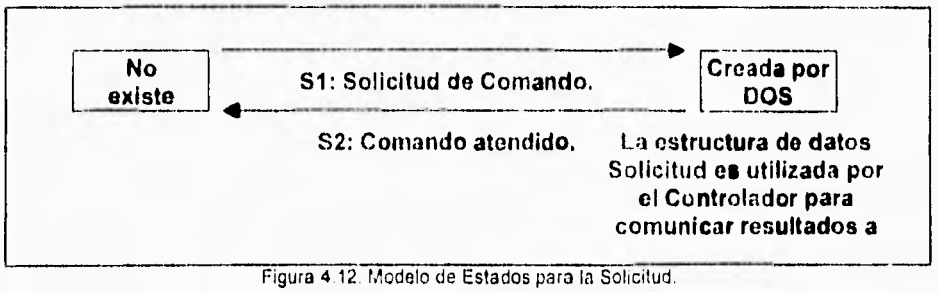



Figura 4.12. Modelo de Estados para la Solicitud.

**Cada vez que DOS requiere invocar un Comando del Controlador crea una instancia de la Estructura de datos Solicitud, la cual se utiliza para pasar parámetros al Controlador y conocer el resultado de la operación.**

**El Modelo de Estados para la Entidad Estrategia es el siguiente:**

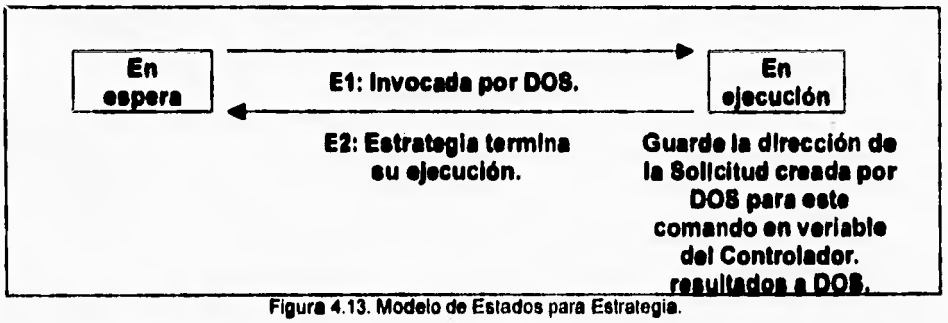



Figura 4.13. Modelo de Estados para Estrategia.

**DOS invoca a la rutina Estrategia inmediatamente después de crear la estructura de datos Solicitud. La rutina Estrategia obtiene la dirección de la Solicitud y la almacena en una variable del Controlador, donde puede ser fácilmente accesada por la rutina Interrupción y por las rutinas de Atención a Comandos.**

**El Modelo de Estados para la Entidad Interrupción es el siguiente:**

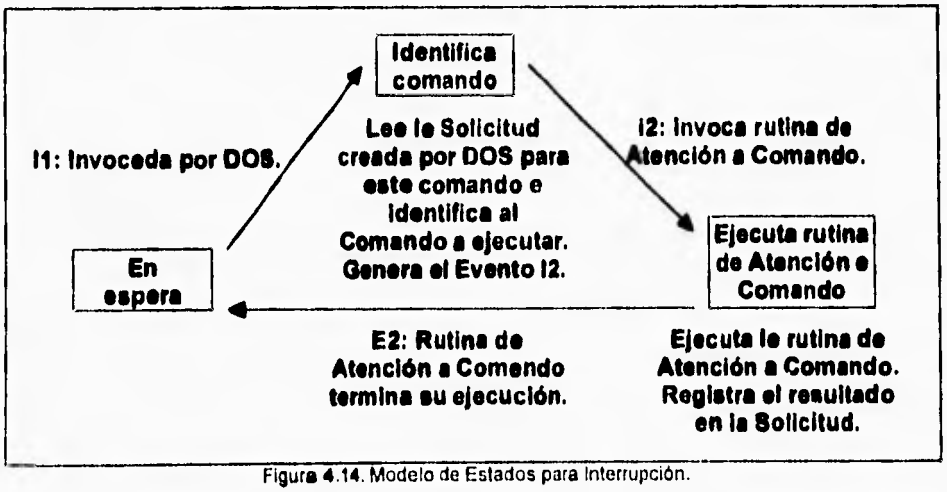



Figura 4.14. Modelo de Estados para Interrupción.

La rutina Interrupción es invocada por DOS después de la invocación a la rutina Estrategia. Localiza a la Solicitud a partir de la dirección obtenida por la rutina Estrategia y lee de ella el código numérico del comando solicitado. Con ésto obtiene la dirección de la rutina que atiende al comando y la llama.

El Modelo de Estados para la Entidad Rutina de Atención a Comando es el siguiente:

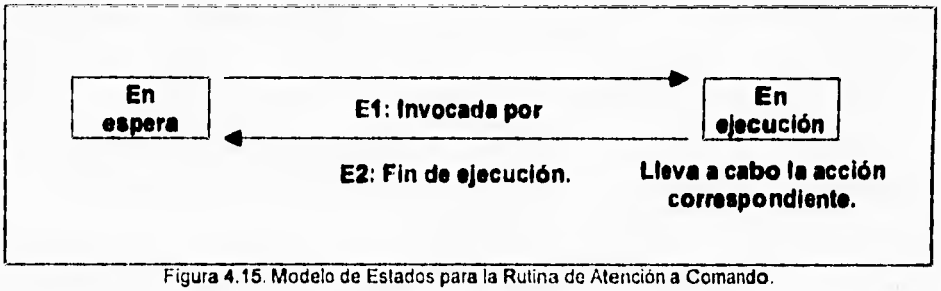



Figura 4.15. Modelo de Estados para la Rutina de Atención a Comando.

La Rutina de Atención a Comando es invocada por la rutina Interrupción para realizar el procesamiento especifico de un comando ordenado por DOS. Es la que realmente realiza el trabajo pesado en un Controlador, y en gran parte la que distingue a un Controlador de dispositivos de otro puesto que realiza la interfaz directa al dispositivo controlado. Existe una instancia por cada comando que el Controlador implemente.

Después de desarrollar los Modelos de Información y de Estados la mayor parte del código del sistema queda distribuida y explicada a través de las acciones descritas en los Modelos de Estados de cada Entidad, y en ellas se basa la siguiente etapa.
Para sistemas de carácter menos técnico dicha etapa se desarrolla durante la fase de Análisis, pero a medida que el sistema tiende a ser menos convencional o más técnico van surgiendo factores cuyas causas originarias y efectos sobre las otras fases del desarrollo pueden ser difíciles (si no es que imposibles) de cuantificar. Dichas causas y efectos se conocen y manejan conforme se diseña el resto del Sistema, pues el hacer lo contrario puede inducir mayor complejidad y pérdida de generalidad sobre el análisis de un Sistema. Teniendo en mente esta situación, se decidió postergar hasta la fase de Diseño el análisis preciso de cuales serian los comandos del Controlador que se implementarian, así como cual seria su comportamiento. Para lograr facilidad de comprensión, se pretendió conservar el Reporte de la fase de Análisis tan sencillo y breve como fue posible. La siguiente sección presenta el Reporte de la fase de Diseño del Controlador.

# Reporte del Diseño.

## Definición de la plataforma de desarrollo.

### Hardware.

Respecto al hardware requerido, el proyecto no es exigente. Basta con una computadora PC-AT (Procesador 80286, Aunque el desarrollo obviamente se eficientiza con un procesador más avanzado), un disco duro y una unidad para disco flexible. Las demás caracteristicas no son determinantes, aunque obviamente algunas de ellas podrían ser útiles. Por ejemplo, entre más rápido sea el procesador más dinámico será el desarrollo; si se tiene la suficiente memoria RAM extendida se puede crear un disco virtual con el que se de velocidad a las compilaciones. Por otro lado, es recomendable respaldar el disco duro usado durante la etapa de desarrollo, pues el desarrollo de accesos a disco duro a nivel BIOS puede arriesgar la integridad de la información contenida.

### Utilerías de desarrollo.

- **IntrSpy.** En este rengión merece ser mencionado de manera especial el depurador de bajo nivel IntrSpy, el cual es particularmente adecuado para este tipo de desarrollos de software y en sus primeras versiones fue un producto Shareware. Entre las principales caracteristicas de éste está el ser un depurador orientado a eventos (Es decir, se especializa en la detección y seguimiento -Tracing- de interrupciones activadas por eventos de hardware como intento de escritura en un disco dañado o de software como peticiones de salida a un dispositivo). Por otro lado, posee un lenguaje tipo Script integrado. Por lo demás, debido a la naturaleza tan caracteristica de los controladores de dispositivos, es casi imposible user herramientas sofisticadas de depuración de código, así como conseguir las pocas que sean adecuadas (IntrSpy fue conseguido como Shareware). Sin embargo, si puede ser muy útil una aplicación que permita explorar de manera sencilla el contenido de la memoria RAM cuando se tiene algún error. En este sentido dos programas fueron muy útiles durante las fases de pruebas, de diseño y de desarrollo. Estas son:
- **Debug.** Herramienta de depuración contenida en el sistema operativo DOS, digna de nombrarse por su concepción modesta pero de gran consistencia y flexibilidad.
- **ProView.** Herramienta de exploración de memoria RAM de McAfee, la cual por estar en su primera versión (por lo menos en el momento de este desarrollo) no tiene gran soporte para la exploración de Controladores de dispositivos aunque si tiene opciones específicas para localizarlos y verlos en pantalla. Por otro lado, posee una interfaz al usuario sumamente amigable (a diferencia de otros productos de McAfee) y permite mostrar información de memoria RAM en tiempo real. En un futuro puede ser la mejor opción para este tipo de desarrollos.

## Otras herramientas de desarrollo.

**Por su naturaleza, Este tipo de aplicaciones es comúnmente escrito en lenguajes de bajo o mediano nivel (Esto debido a requerirse código compacto que permita controlar el Hardware del equipo de manera eficiente). Inicialmente se consideró desarrollar el código del proyecto completamente en uno de dos lenguajes: Ensamblador o lenguaje "C". Se realizaron pruebas consistentes en la escritura de pequeños controladores en cada una de estas plataformas, obteniendose las conclusiones siguientes:**

### 1. Para el desarrollo con Lenguaje Ensamblador:

- **Ventajas.**
  1. **El código resultante es extremadamente compacto, lo cual ayuda a lograr ahorros considerables de memoria RAM cuando el controlador ya está instalado (Las pruebas realizadas permitieron escribir Controladores de Dispositivos relativamente sencillos -dispositivos nulos- en Lenguaje Ensamblador en solo 100 bytes de tamaño). Este criterio es de gran peso, pues todo Controlador de Dispositivos debe residir en memoria RAM y por tanto será mejor entre menor sea su tamaño.**
  2. **Se logra un control completo sobre el código y se evitan dependencias que se derivan del código generado por un compilador comercial, por ejemplo en lo referente a:**
     1. **El ordenamiento de los grupos de un programa.**
     2. **Variables "fantasma" generadas para alinear en direcciones par las estructuras de datos compuestas.**
     3. **Eliminación de instruccionas superfluas.**
  3. **El código fuente refleja más claramente la estructura que un controlador debe poseer.**
  4. **Control completo sobre las llamadas a Rutinas de Interrupción. Con esto se puede evitar las llamadas a Interrupciones que ejecuten código de DOS (El cual no se debe ejecutar en un controlador de dispositivos).**
  5. **Independencia de Productos de Desarrollo. Es decir, El código resultante se puede ensamblar con un producto de cualquier marca y el resultado será siempre el mismo.**

- **Desventajas.**
  1. **Se requiere un nivel considerablemente mayor de conocimientos sobre la arquitectura de la computadora y sobre el lenguaje ensamblador.**
  2. **Se debe iniciar el código deade cero. En otras palabras, no se puede aprovechar el código que proveen las librerias de un compilador (A menos que se conozcan con exactitud los formatos de llamada a sus rutinas). Por**

ejemplo, no se pueden hacer llamadas a rutinas de manejo de cadenas ya disponibles en el compilador de Lenguaje C.
3. Los códigos fuente resultantes son más difíciles de depurar, debido al mayor tamaño del programa.
4. El tiempo ocupado en un ciclo de desarrollo es mayor que si se usa un lenguaje de mediano nivel, aún cuando se mantenga una buena disciplina de desarrollo.

## Para el desarrollo con lenguaje C:

### Ventajas.

1. El código es mucho más fácil de depurar o mantener que el desarrollado en ensamblador.
2. Los tiempos de desarrollo se acortan.
3. Se puede aprovechar el código ya implementado como parte del lenguaje (como el código de operaciones aritméticas), siempre y cuando se esté seguro de que no se hacen llamadas a DOS riesgosas para la integridad del Controlador de dispositivos.

### Desventajas.

1. Se deben realizar muchas adecuaciones sobre el código fuente para ajustarlo a la estructura requerida por un controlador de dispositivos. Ésto se debe a que un Controlador de Dispositivos demanda una estructura rígida para poder funcionar correctamente, y un compilador de Lenguaje C no impone una estructura que sea muy fácil de adaptar a los requerimientos que en este caso se tienen.
2. La estructura del controlador no es directamente aparente desde el código fuente en Lenguaje C, debido a que se requiere realizar manipulaciones de código para lograrlo.
3. Un compilador comercial, debido a su generalidad, produce más código que el que en ciertos casos produciría un Ensamblador. Por tanto el Controlador resultará más grande.
4. Si el código generado por el compilador de Lenguaje C invoca alguna interrupción de DOS el controlador puede fallar inadvertidamente, debido a la naturaleza no Reentrante de DOS. A este respecto se debe hacer notar que, estrictamente, no todas las llamadas a DOS violan la condición de No Reentrancia del Sistema Operativo. Sin embargo, en el momento de realizar este trabajo no se disponía de documentación formal que indicara cuales llamadas a DOS se pueden invocar desde BIOS o desde un Controlador de Dispositivos (De hecho tal vez esa información no esté disponible actualmente al público).
5. Las dependencias del compilador pueden causar errores muy difíciles de depurar. Por ejemplo, al declarar estructuras de datos compuestas algunos

compiladores tratan de normalizar el tamaño de esas variables compuestas a valores pares y para lograr ésto agregan bytes "fantasma" dentro de la variable compuiesta.

6. Compiladores de diferentes marcas (o incluso versiones) pueden generar código diferente, que en algunos casos provoqua errores de ejecución y en otros no. Es decir, el código no es fácilmente portable entre diferentes compiladores.

### Se concluyó lo siguiente:

Las pruebas realizadas con Lenguaje Ensamblador mostraban que el desarrollo de un Controlador de cierta dificultad iba a resultar muy lento, mientras que las realizadas con lenguaje C fueron muy difíciles de implementar debido a que las múltiples dependencias del compilador provocaban errores de difícil depuración y conforme el programa crecía estos problemas se multiplicaban.

Considerando toda esta información, se optó por hacer un desarrollo mixto, el cual tratara de aplicar lo mejor de ambos lenguajes en las partes en que más adecuado resultara. De esta forma se lograron las siguientes características:

1. El programa resultante refleja de manera clara la estructura del Controlador, de tal forma que es más fácil la comprensión del código.
2. El tiempo ocupado en el desarrollo y la complejidad del código resultante son drásticamente menores a los de los 2 desarrollos anteriores. Ésto es más notable conforme se incrementa la complejidad del proyecto.
3. Evitando llamadas a rutinas de libreria riesgosas no existe peligro de llamadas a interrupciones DOS desde el código generado.
4. No se implementa cada función desde cero y el código resultante es más fácilmente comprensible y depurable.

## Flujo de información externo (DOS a Controlador).

Desde el punto de vista de su situación con respecto al Sistema Operativo de un equipo (Cualquiera que éste sea, no necesariamente una PC) y con respecto al usuario final, Un Controlador de dispositivos no es un programa tradicional. Ésto se debe a que un controlador de dispositivos no se "monta" sobre el Sistema Operativo para trabajar. Por el contrario, forma parte integral de éste (Como ya se vio en la sección dedicada a historia, un Controlador de dispositivos puede integrarse al Sistema operativo o, como en los equipos más antiguos, literalmente formar parte de el) y convive con el cumpliendo las reglas que le imponga. Gráficamente, La siguiente ilustración muestra ésto para el caso DOS:

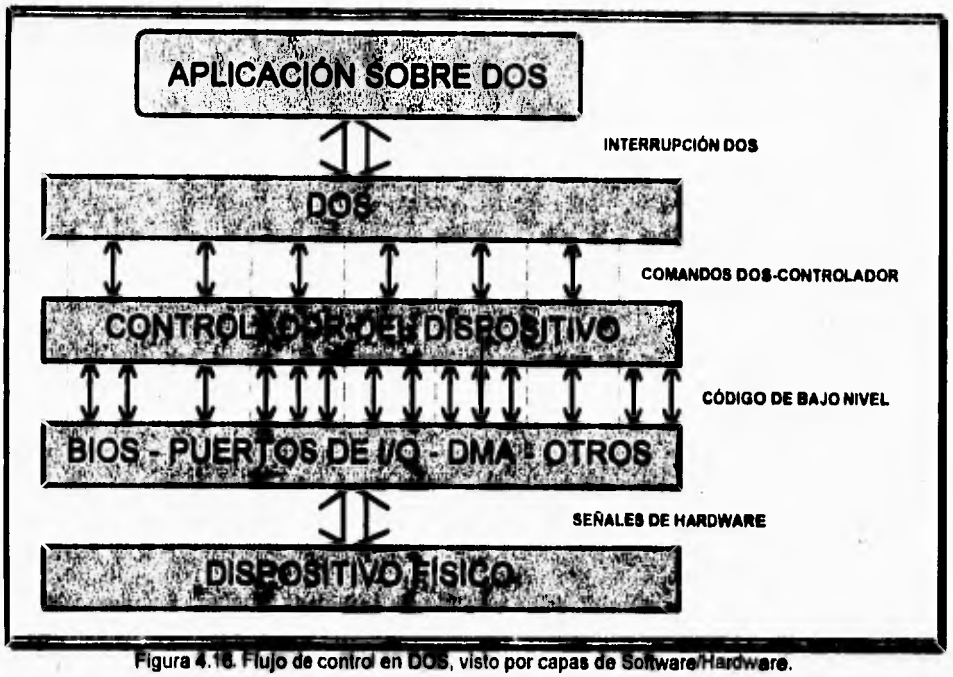



Figura 4.16. Flujo de control en DOS, visto por capas de Software/Hardware.

**En la figura anterior se encuentra la clave para definir el flujo de control externo válido para todo Controlador de dispositivos en DOS. Básicamente se aprecia que la aplicación que se ejecuta sobre la capa de software proporcionada por el sistema operativo DOS realiza una llamada a una rutina de la API (Application Programming Interface, Interfaz de programación de aplicaciones) de DOS para accesar un dispositivo determinado. A su vez, el código de DOS se encarga de "desdoblar" esa llamada en un conjunto de comandos que envía al Controlador de dispositivos encargado de atender al dispositivo en cuestión. El controlador del dispositivo, a su vez, ejecuta cada uno de los comandos haciendo llamadas al código de BIOS, realizando Entrada/Salida a través de puertos de memoria, usando el estándar DMA (Direct Access Memory, Acceso directo a Memoria usado para trasladar información entre un dispositivo y memoria más rápidamente al evitar que los datos pasen a través del procesador), etcétera. Un ejemplo de este proceso se obtiene al rastrear todos los comandos en que se desmenuza una petición del sistema operativo a un controlador de disco. De tal forma, el conjunto de llamadas obtenido al enviar al sistema operativo un comando de directorio (DIR E:\) fue:**

```
(Handler for calls to 03AA:0E34 was already stopped.)
D:DD.SCR compiled successfully.

---------- Start of Report ----------

01 - media check
02 - build bpb
01 - media check
04 - input
01 - media check
02 - build bpb
04 - input
04 - input
01 - media check
02 - build bpb
04 - input
04 - input
04 - input
04 - input
04 - input
04 - input
04 - input
01 - media check
02 - build bpb

---------- End of Report ----------

All counters zero.

Intrspy returned successful status.
```

El listado anterior fue obtenido usando el depurador IntrSpy, y se resumió para simplificarlo (El listado original mostraba 97 llamadas al controlador).

Como se puede ver, DOS cumple su papel de simplificar y generalizar la interfaz de programación de un sistema tradicional a través del uso de los controladores de dispositivos.
Por otra parte, es importante recordar que cada llamada a un comando desde DOS de desglosa en realidad en dos llamadas: una a la rutina **ESTRATEGIA** y otra a la rutina **INTERRUPCIÓN**. Del capítulo dedicado a los controladores de dispositivos en DOS sabemos que la única función de la rutina ESTRATEGIA en DOS es la de obtener la dirección del Request Header para ser usado por la rutina INTERRUPCIÓN. El objetivo de ésta última, a su vez, es el de invocar a la rutina

**que atienda a un comando determinado. Desde este punto de vista, el siguiente diagrama de flujo de Control (Figura 4.3) muestra la secuencia en que ambas rutinas se llaman y las acciones que realizan:**

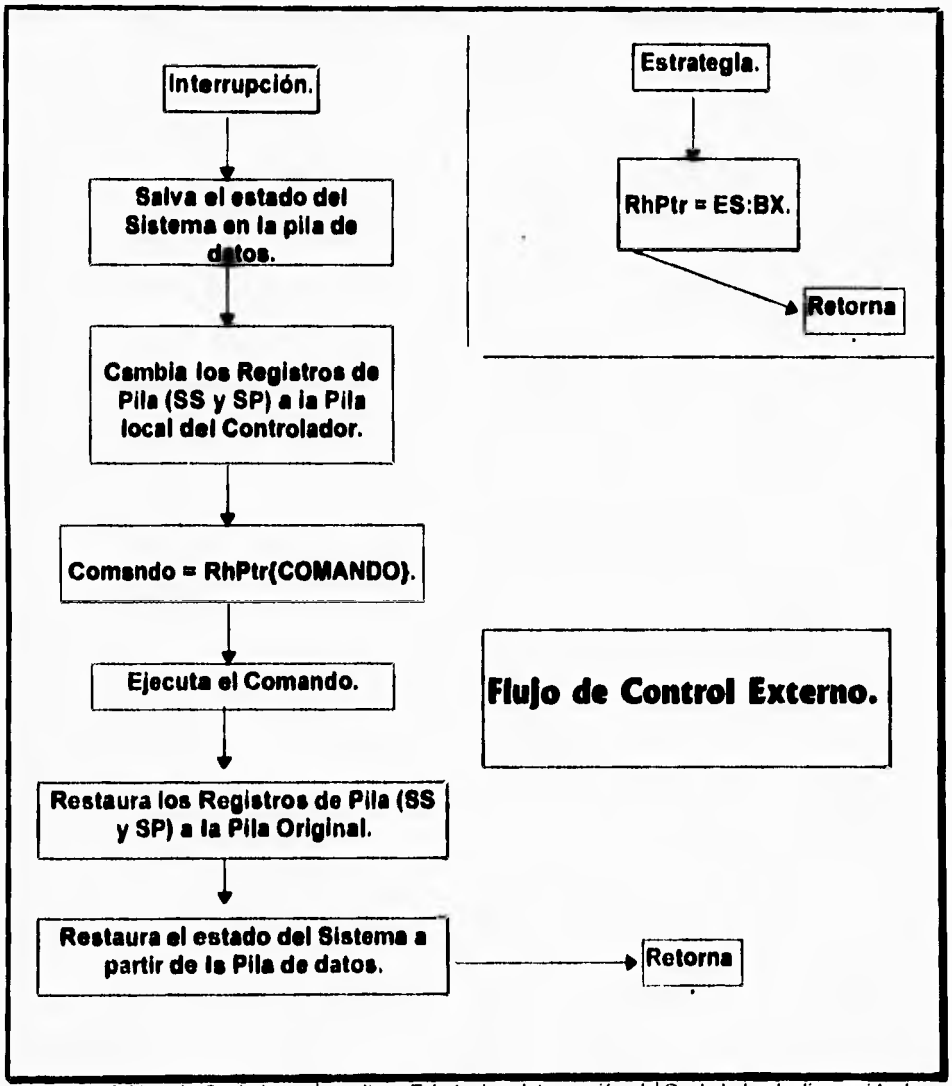



Figura 4.17. Flujo de Control para las rutinas *Estrategia* e *Interrupción* del Controlador de discos virtuales.

**El diagrama de flujo de información que indica las estructuras de datos que las dos rutinas del controlador accesan se muestra:**

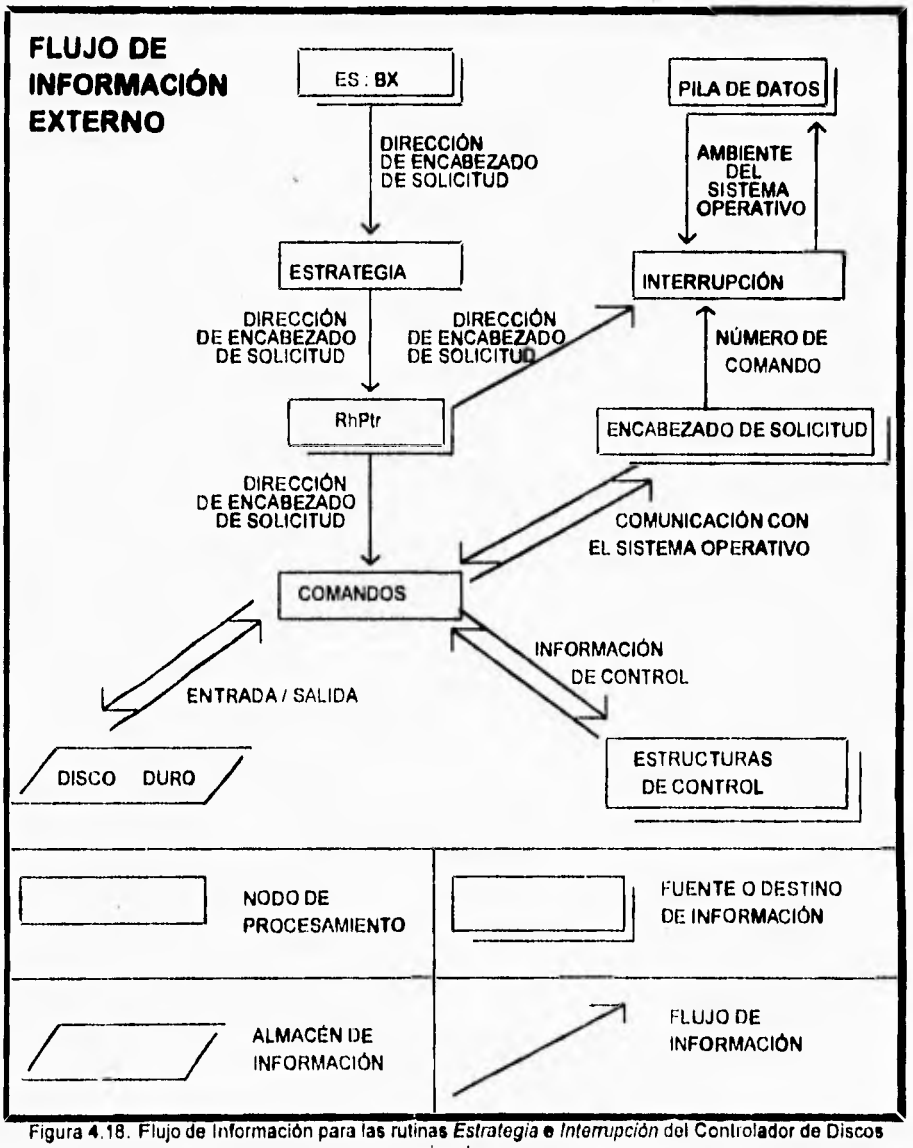



Figura 4.18. Flujo de Información para las rutinas *Estrategia* e *Interrupción* del Controlador de Discos virtuales.

Es importante hacer notar que la cantidad y tipo de comandos que DOS envia al controlador no es siempre el mismo para una determinada petición de aplicación. Ésto se encuentra influido por factores como la configuración de DOS y por la cantidad de buffers libres en ese momento pero principalmente por las propiedades que se hayan dado al controlador en el momento de su diseño. La definición de estas propiedades es vital para la vida útil de un controlador de dispositivos y es el punto central del siguiente párrafo.

## Determinación de las características del controlador de disco.

Como se comentó en el párrafo anterior, las caracteristicas que desde la fase de diseño se den a un Controlador de Dispositivos son vitales para su comportamiento durante la etapa de operación. La sección en la que se definen las caracteristicas de un controlador es su encabezado, y en particular el caracter de "Atributo". Como ya se vio anteriormente, el formato del encabezado (representado en lenguaje C) es el siguiente:

```
typedef    struct device_haader_struct {
      struct device_header_struct    far * sigdev ;
      unsigned int                        atributo ;
      void                                (* estrategia)  (void) ;
      void                                (* interrupcion) (void) ;
      unsigned char                      nombre [8] ;
} deviceheader_t ;
```

Para nuestro controlador los valores asignados a cada elemento son los siguientes:

```
sigdev = -1.
atributo = 2000 Hexadecimal.
estrategia = _estrategia.
interrupcion = _interrupcion.
nombre = ASCII (5) + "     ".
```

Las significados de estos valores son los siguientes:

A los 4 digitos que conforman a la variablesigdev se les asigna un -1 para indicar que no hay otro controlador en el archivo que contiene a nuestro programa. Si nuestro archivo contuviera el código de más de un Controlador esta variable apuntaria al encabezado del siguiente Controlador dentro del archivo. Posteriormente, cuando el controlsdor ya esté instalado en la memoria RAM del sistema, DOS se encargará de asignarle un valor adecuado, el cual indicará el

inicio del siguiente controlador dentro de una lista simplemente ligada que el Sistema operativo se encarga de administrar. A esto se debe que se utilice un apuntador de tipo FAR, pues el Controlador podría estar en prácticamente cualquier sitio en la memoria RAM.

Los apuntadores **atributo** y **estrategia** contienen las direcciones de inicio de las rutinas correspondientes del mismo nombre dentro del código del Controlador. Vease que ambos apuntadores son de tipo NEAR (definido implícitamente), pues solo indican un offset (Corrimiento) a partir del inicio del controlador.

la cadena **nombre** indica en su primer caracter el número máximo de dispositivos que el controlador deberá soportar. Se puso en 5 pero se puede redefinir posteriormente, durante el comando de inicialización. El resto de la cadena se deja en blanco (Si el dispositivo fuera de caracter esta variable se habría usado para guardar el nombre del dispositivo, como en COM1:, LPT1:, etcétera).

La variable **atributo*** es la más importante pues determina las características y comportamiento del Controlador. El valor que se le asignó tiene las siguientes implicaciones para el controlador:

- **Primer dígito hexadecimal:** El controlador maneja uno o más dispositivos de bloque, y no soporta control de E/S por el sistema. Esto último implica que no tiene implementada la respuesta a los comandos de control de Entrada/Salida (comandos 3 y 12, comandos IOCTL). Por otro lado, el dispositivo no es tipo IBM, lo cual significa que DOS no deberá hacer ciertas suposiciones al enviarle comandos y deberá siempre tratarlo como un dispositivo no estándar.
- **Segundo dígito hexadecimal:** El dispositivo nunca cambia durante el uso normal del sistema. En otras palabras, nuestro disco virtual simula un disco duro y no un disco flexible. Técnicamente hablando ésto implica que el controlador no necesitará implementar (Pues DOS nunca los invocará) los comandos 13, 14 y 15 (Comandos "Device Open", "Device Close" y "Removable Media"), los cuales son utilizados para permitir al desarrollador mejor control de medios removibles, como discos flexibles, cintas de respaldo, etcétera.
- **Tercer dígito hexadecimal:** El controlador no implementará las respuestas a los comandos 19 ("Generic IOCTL") ni 25 ("IOCTL Query"). En consecuencia, El controlador no se apegará al estándar de control de E/S definido por el Sistema Operativo, ni aceptará interrogaciones acerca de sus atributos de E/S genérica. ambos comandos están estrechamente relacionados, pero su uso en la actualidad no está difundido.

---

* Esta sección discute solo los valores de atributo asignados al Controlador presentado en este trabajo. Si desea más información acerca de la definición de los atributos de un controlador y de los comandos de controls de Entrada/Salida referidos consulte el capítulo dedicado a la definición de la estructura de los Controladores de Dispositivos ("Controladores de dispositivos en DOS.").

- **Cuarto dígito hexadecimal: Su valor indica que el controlador no reemplaza a ninguno de los controladores estándar definidos por DOS para los dispositivos de Entrada estándar, Salida estándar y Reloj (STDIN:, STDOUT:, CLOCK$:). Ésto es claro, puesto que el controlador está agregando el control de un nuevo dispositivo, no redefiniendo el comportamiento de uno ya existente.**

**Como se acaba de ver, la determinación de las características de un controlador en la fase de diseño influye en gran parte sobre los comandos a los que el controlador deberá responder. Por tanto, y dado que ya se cuenta con la información suficiente para ello, Nuestro siguiente paso consistirá en determinar los comandos que nuestro controlador deberá implementar y posteriormente describir su función en relación con el dispositivo.**

## Selección de los comandos a implementar.

**La siguiente tabla muestra los comandos que el controlador deberá implementar, de acuerdo a la definición previa de sus atributos. La primera columna muestra el número (con que DOS lo identifica) y el nombre del comando (indicado en inglés para mayor brevedad). La segunda columna indica los comandos que DOS es capaz de enviar a todo controlador de bloque (Si un comando se usa para dispositivos de bloque se identifica con una letra$ en este campo). Vease que el hecho de que un comando tenga este identificador no implica que sea exclusivamente de tipo Bloque. De entre los seleccionados en la segunda columna, en la tercera columna se indican aquellos que si se implementarán para nuestro controlador. por último, en la cuarta columna se indica un código identificando el motivo por el que un comando se implementa o no, para todos los renglones de la tabla. Los diferentes códigos numéricos que se aplican a la columna de motivos se explican a continuación.**

## Explicación de Códigos para la cuarta columna:

**1: En todo controlador se implementa.**
**2: En todo controlador de bloque se implementa.**
**3: No aplica por los atributos asignados al controlador.**
**4: No aplica por ser solo para dispositivos de tipo caracter.**
**5: Comando no definido.**

**Para mayor claridad se utiliza un marco delimitando algunas áreas de la tabla para indicar los comandos que son aplicables a los controladores de bloque. Partiendo del subconjunto formado por este marco se crea un subconjunto aún menor al sombrear los comandos que nuestro controlador deberá implementar.**

| NOMBRE | APLICA | PASA | MOTIVO |
|---|---|---|---|
| 0-Inicialization | S | S | 1 |
| 1-Media Check | S | S | 2 |
| 2-Get BPB | S | S | 2 |
| 3-IOCTL Input | S | | 3 |
| 4-Input | S | S | 1 |
| 5-Nondestructive Input | | | 4 |
| 6-Input Status | | | 4 |
| 7-Input Flush | | | 4 |
| 8-Output | S | S | 1 |
| 9-Output with Verify | S | S | 2 |
| 10-Output Status | | | 4 |
| 11-Output Flush | | | 4 |
| 12-IOCTL Output | S | | 3 |
| 13-Device Open | S | | 3 |
| 14-Device Close | S | | 3 |
| 15-Removable Media | S | | 3 |
| 16-Output Til Busy | | | 4 |
| 17-No definido | | | 5 |
| 18-No definido | | | 5 |
| 19-Generic IOCTL | S | | 3 |
| 20-No definido | | | 5 |
| 21-No definido | | | 5 |
| 22-No definido | | | 5 |
| 23-Get Logical Device | S | | 3 |
| 24-Set Logical Device | S | | 3 |
| 25-IOCTL Query | S | | 3 |

Tabla 4.1. Comandos a implementar en el Controlador de Discos virtuales.

**Una vez inferido el conjunto de comandos a los que el controlador debe responder, en la siguiente sección se indicará de manera detallada cual debe ser la conducta de cada uno de ellos.**

## Flujo de información interno (por comandos).

En esta sección se describirá detalladamente el flujo de información que se da para cada uno de los comandos que se implementan en el controlador. Para obtener información acerca de la función genérica de cada comando dentro de un controlador consulte el capítulo dedicado a los controladores de dispositivos en DOS.

### Comando 0: Inicialization (Inicialización del Controlador).

Para comprender mejor el trabajo de este comando es necesario recordar que el controlador debe iniciar teniendo el nombre de por lo menos un archivo que ya exista en el disco duro, o recibir éste durante la inicialización. Por otro lado, para simplificar el desarrollo del proyecto se inició considerando las siguientes simplificaciones:

1. El controlador solo trabaja sobre la primera partición del primer disco duro de la computadora. Ésto implica que dicha partición debe contener un sistema de archivos DOS.
2. El archivo que represente al controlador deberá estar siempre en la raiz del sistema de archivos sobre el que se trabaje.

Estas restricciones se pueden eliminar de manera fácil en versiones sucesivas.
Una vez tomados en cuenta los factores descritos, el sentido común nos indica que el controlador debiera empezar por conocer las caracteristicas del disco fisico sobre el que trabajará. después de ésto debiera conocer la ubicación y caracteristicas del disco lógico (o archivo) que usará. Posteriormente inicalizaría todas aquellas entidades que describan el ambiente sobre el que estará operando. De manera detallada, las acciones que este comando realiza son:

1. Lee las áreas de partición y arranque del disco fisico y calcula valores de caracter general que le servirán en un futuro. Almacena la información leida y calculada en una entidad llamada File_System que representa al sistema de archivos en general. Si se presenta algún error en este momento el controlador aborta su instalación.
2. Lee la linea de comando que se proporcione después de la declaración del controlador dentro del archivo config.sys. Esta linea deberá indicar al controlador cuantos discos virtuales controlará y cuales son los nombres de los archivos que los contienen. El formato de esta linea de comandos se especificó en la fase de análisis. Como consecuencia de alguna inconsistencia no solucionable en esta linea de comandos el controlador aborta su instalación. Si su interpretación fue correcta, El controlador muestra en pantalla su configuración inicial. Internamente, el resultado de la interpretación se almacena en una entidad llamada **Volúmenes**, cuyo propósito es representar a todos los dispositivos virtuales que el controlador maneje.
3. Para cada dispositivo, se busca en el directorio raiz el archivo que le corresponda y se genera (a partir de la primera tabla de FAT del disco fisico) un vector que contiene la cadena de clusters que conforman al archivo. Esta

información se registra en la entidad Volúmenes. A continuación se lee el primer sector del primer cluster del archivo y se toma como el sector de arranque del disco virtual en turno. A partir de esta lectura se forma el BPB (BIOS Parameter Block) y se guarda en una entidad llamada bpbs_arreglo y que es requerida por la especificación estándar de controladores en DOS. Cualquier falla durante todo este proceso (El archivo no existe, la cadena FAT es extremadamente grande o inconsistente, errores de lectura, etcétera) provoca que el controlador aborte.

4. Retorna a DOS (A través de la entidad que representa al Request Header) la dirección del vector de BPB's, el total de dispositivos que el controlador manejará, la dirección a partir de la que se puede desechar el código cuando la inicialización haya terminado (Lo cual ayuda a ahorrar memoria), y el código de salida de este comando.

La figura 4.5 muestra el flujo de control que representa los puntos descritos:

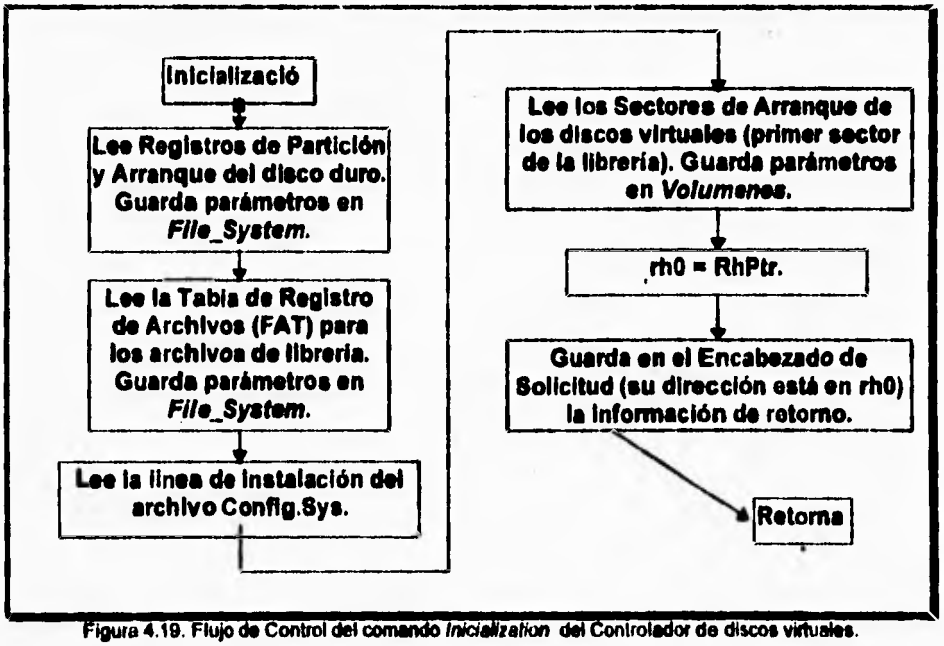



Figura 4.19. Flujo de Control del comando *Inicialization* del Controlador de discos virtuales.

El diagrama de flujo de información en la figura 4.6 muestra las estructuras de datos que este comando utiliza.

## Comando 1: Media Check (Checa si el medio está presente).

**Si consideramos que la función del comando Media Check es hacer saber al sistema operativo si el dispositivo ha cambiado (Como cuando se cambia un disco flexible por otro), entonces la implementación de este comando es sumamente sencilla pues en nuestro proyecto de Controlador el disco que representa a un dispositivo nunca cambia durante la operación del sistema, sino hasta que se modifica el archivo Config.Sys y se reinicia la computadora.**

1. **Se obtiene un apuntador a la entidad que representa al Request Header y se retorna en esta el aviso de no cambio de disco y el código de terminación.**

**El diagrama de flujo de control en la figura 4.7 muestra este proceso.**

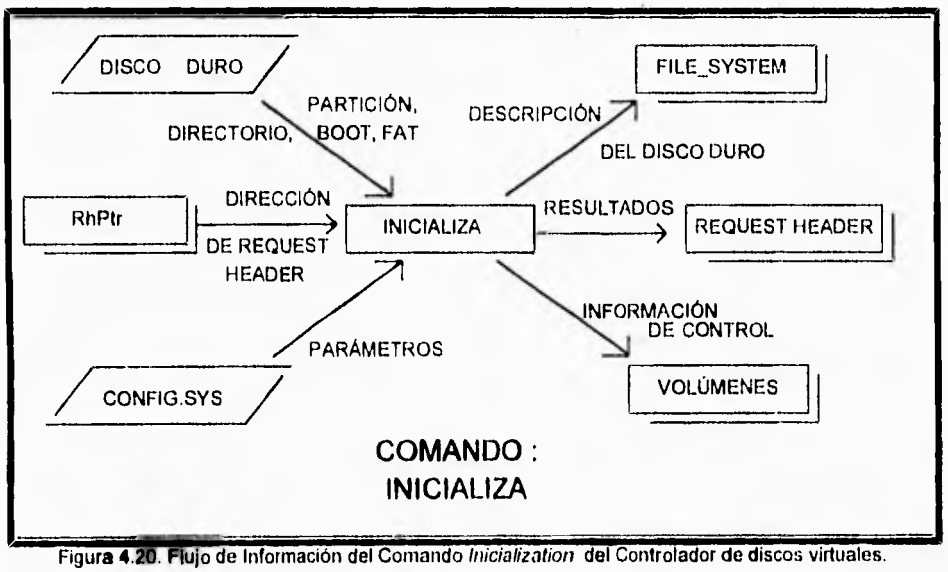



Figura 4.20. Flujo de Información del Comando *Inicialization* del Controlador de discos virtuales.

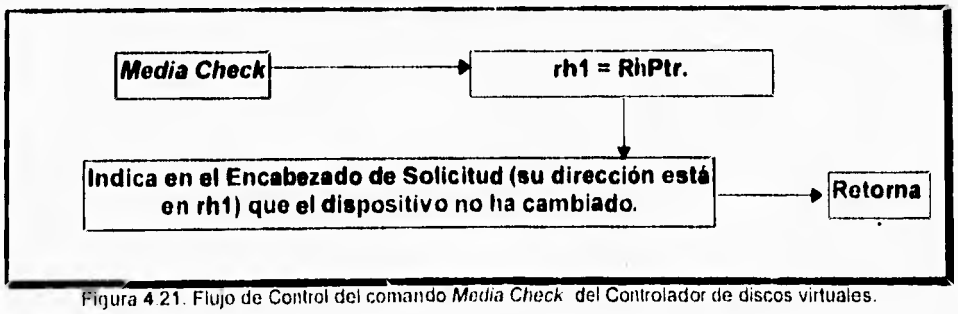



Figura 4.21. Flujo de Control del comando *Media Check* del Controlador de discos virtuales.

**En el siguiente diagrama de flujo de información se muestran las variables utilizadas por el comando *Media Check*:**

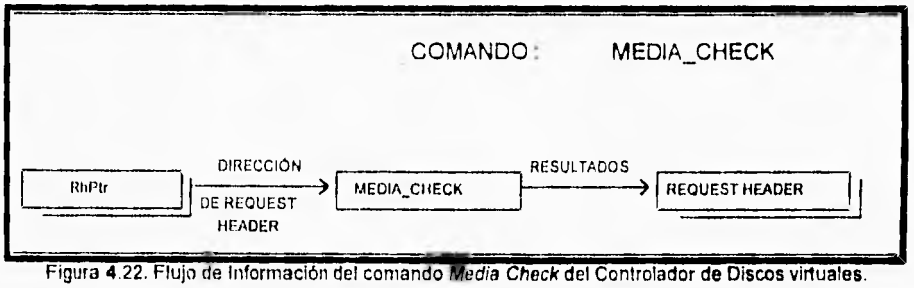



Figura 4.22. Flujo de Información del comando *Media Check* del Controlador de Discos virtuales.

## Comando 2: Get BPB (Retorna Bloque de Parámetros de BIOS).

**El objetivo de este comando es retornar a DOS el BPB (BIOS Parameter Block, Bloque de parámetros de BIOS) del dispositivo que se solicite, así como la dirección del vector de BPBs. Éste es un proceso común en DOS y ya fue realizado por lo menos una vez durante la etapa de inicialización. por tanto el proceso es el mismo:**

1. **Se lee el primer sector del primer cluster del archivo y se toma como el sector de arranque del dispositivo. A partir de esta lectura se forma el BPB y se guarda en bpbs_arreglo.**
2. **Se Obtiene un apuntador al Request Header y se retorna en éste el BPB solicitado, la dirección del vector de BPB's y el código de salida de este comando.**

**La figura 4.9 muestra el flujo de control que representa los puntos descritos:**

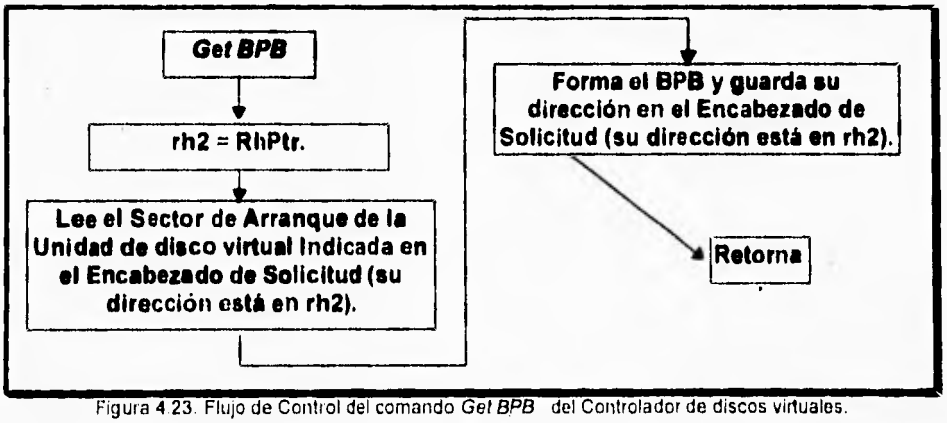



Figura 4.23. Flujo de Control del comando *Get BPB* del Controlador de discos virtuales.

**El siguiente diagrama de flujo de información (figura 4.10) muestra las estructuras de datos que este comando utiliza:**

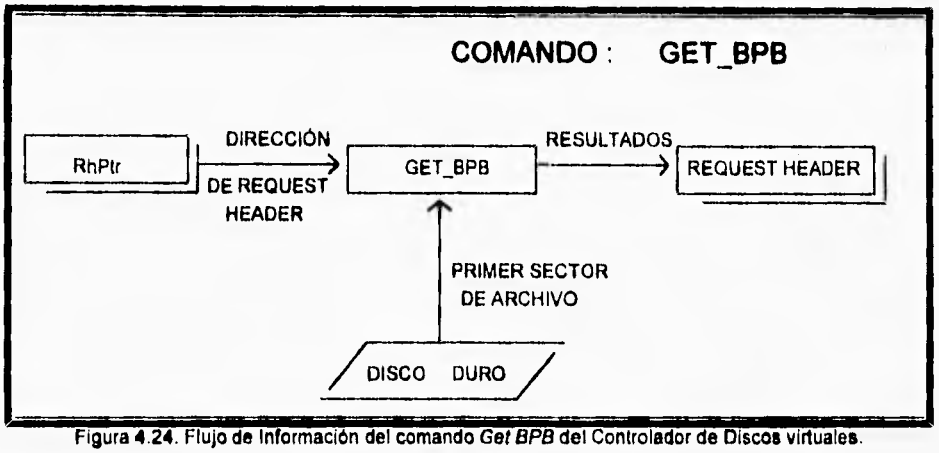



Figura 4.24. Flujo de Información del comando *Get BPB* del Controlador de Discos virtuales.

## Comando 4: Input (Entrada desde el dispositivo).

**Este comando tiene como propósito enviar información desde el dispositivo hacia DOS y su implementación es central al funcionamiento del controlador. Su operación es la siguiente:**

1. **Obtiene del Encabezado de Solicitud (Request Header) el número del sector desde donde se debe iniciar la lectura, cuantos sectores se deben leer y la dirección del buffer destino.**
2. **Recorre la cadena de Celdas que guardan direcciones de FAT (Tabla de registro de Archivos) correspondiente a la unidad de la que se solicita información. Dicha cadena se encuentra en la entidad Volúmenes. Mediante este recorrido se localiza el cluster (Grupo de sectores) de disco real a partir del cual se deberá empezar a copiar información del disco físico hacia el buffer. Posteriormente sigue recorriendo la cadena de FAT a la vez que lee sectores de disco físico, se detiene hasta que logra la cantidad de sectores solicitada. Finalmente, retorna el código de terminación en el Request Header.**

**El siguiente diagrama de flujo de información (figura 4.11) muestra las estructuras de datos que este comando utiliza:**

**El siguiente diagrama de flujo de información (figura 4.10) muestra las estructuras de datos que este comando utiliza:**

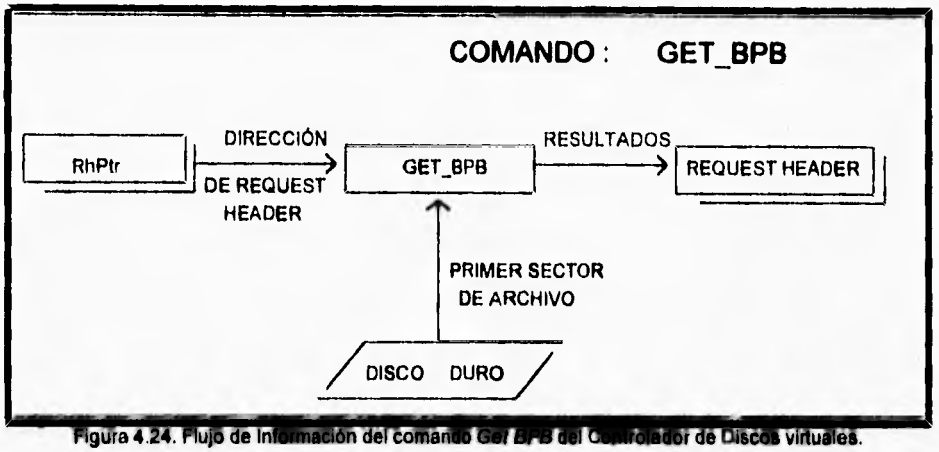



Figura 4.24. Flujo de información del comando *Get BPB* del Controlador de Discos virtuales.

## Comando 4: Input (Entrada desde el dispositivo).

**Este comando tiene como propósito enviar información desde el dispositivo hacia DOS y su implementación es central al funcionamiento del controlador. Su operación es la siguiente:**

1. **Obtiene del Encabezado de Solicitud (Request Header) el número del sector desde donde se debe iniciar la lectura, cuantos sectores se deben leer y la dirección del buffer destino.**
2. **Racorre la cadena de Celdas que guardan direcciones de FAT (Tabla de registro de Archivos) correspondiente a la unidad de la que se solicita información. Dicha cadena se ancuentra en la entidad Volúmenes. Mediante este recorrido se localiza el cluster (Grupo de sectores) de disco real a partir del cual se deberá empezar a copiar información del disco fisico hacia el buffer. Posteriormente sigue recorriendo la cadena de FAT a la vez que lee sectores de disco fisico. se detiene hasta que logra la cantidad de sectores aolicitada. Finalmente, retorna el código de terminación en el Request Header.**

**El siguiente diagrama de flujo de información (figura 4.11) muestra las estructuras de datos que este comando utiliza:**

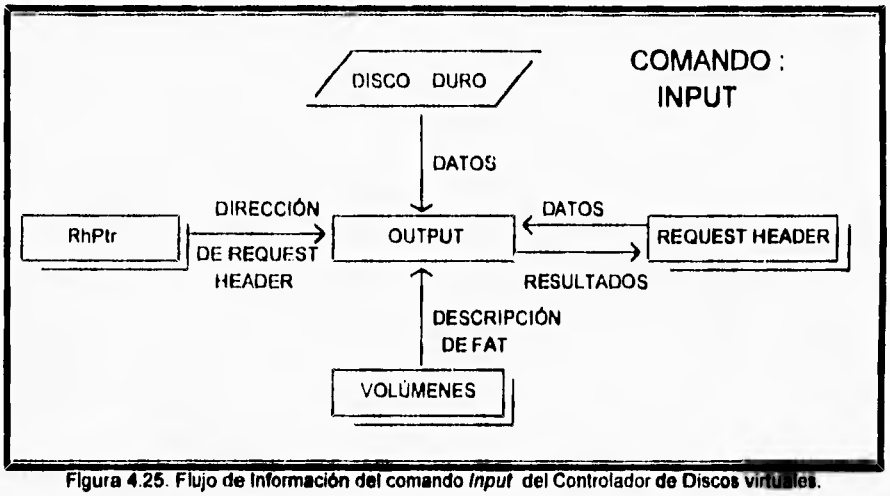



Figura 4.25. Flujo de Información del comando *Input* del Controlador de Discos virtuales.

**La figura 4.12 muestra el flujo de control que representa los puntos descritos:**

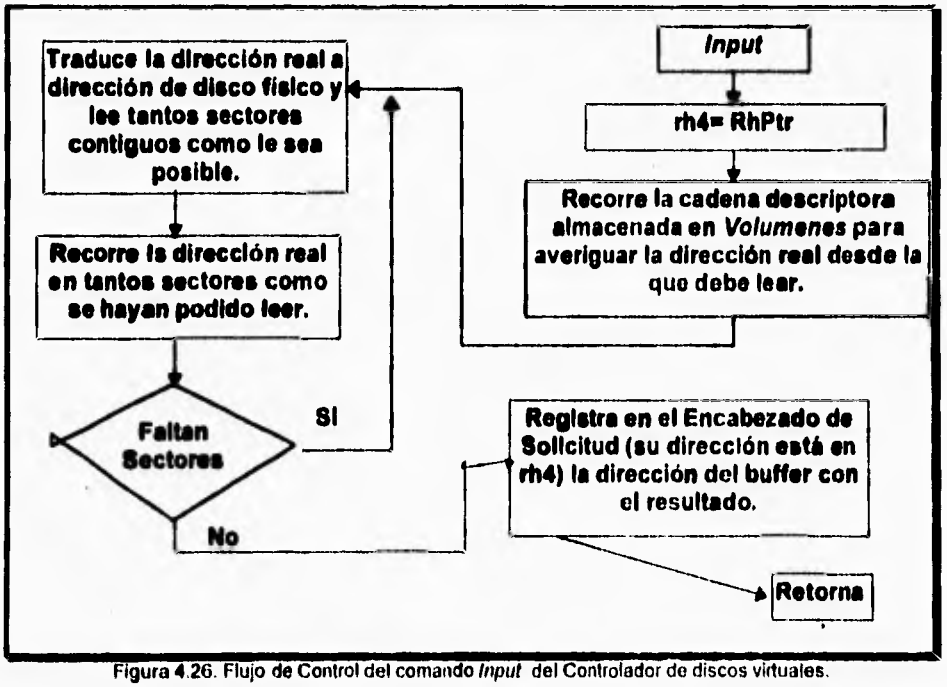



Figura 4.26. Flujo de Control del comando *Input* del Controlador de discos virtuales.

## Comando 8: Output (Salida hacia el dispositivo).

**Su finalidad es la de escribir en el dispositivo virtual la información que para este fin DOS (o la aplicación que lo invoque) le proporcione en un buffer de datos cuya memoria deberá haber sido dispuesta por la aplicación que haga la llamada. Al igual que el comando Input (Entrada desde el dispositivo), su implementación es central para la operación de todo controlador que tenga fines prácticos. En este proyecto trabaja de la siguiente forma:**

1. **Obtiene del Encabezado de Solicitud el número de sector a partir del cual se iniciará la escritura, la cantidad de sectores a escribir y la dirección del buffer en que los datos se encuentran.**
2. **Recorre la cadena de FAT (Tabla de Asignación de archivos) que se encuentra en la entidad Volúmenes para localizar el cluster (grupo de sectores) de disco real a partir del que deberá empezar a copiar información del buffer hacia el disco físico. Después sigue recorriendo la cadena de FAT a la vez que escribe sectores a disco físico (Para poder mapear el disco virtual como si fuera totalmente contiguo, a pesar de que en realidad su representación como un archivo pueda estar fragmentada sobre el disco real). se detiene hasta que logra escribir la cantidad de sectores indicada. Finalmente, retorna el código de terminación en el Request Header.**

**La figura 4.13 muestra el flujo de control que representa los puntos descritos.**

## Comando 9: Output with Verify (Salida hacia el dispositivo con verificación).

**El objetivo de este comando, el último a implementar en nuestro Controlador de discos virtuales, es el de realizar escrituras a dispositivos cuya seguridad de retención de datos se considere baja. La forma en que este comando funciona en la mayoría de sus implementaciones es realizando la escritura solicitada sobre el dispositivo indicado y después leyendo lo previamente escrito para verificar que coincida con el contenido del Buffer inicial. Ejemplos de ésto son ciertos tipos de memoria RAM. Sin embargo éste no es el caso para un disco duro, puesto su seguridad al grabar y recuperar información es alta. Por lo tanto este comando se implementará de forma idéntica al anterior. De hecho, la rutina que atiende a ambos comandos es la misma, por lo tanto no se incluye un diagrama explicativo.**

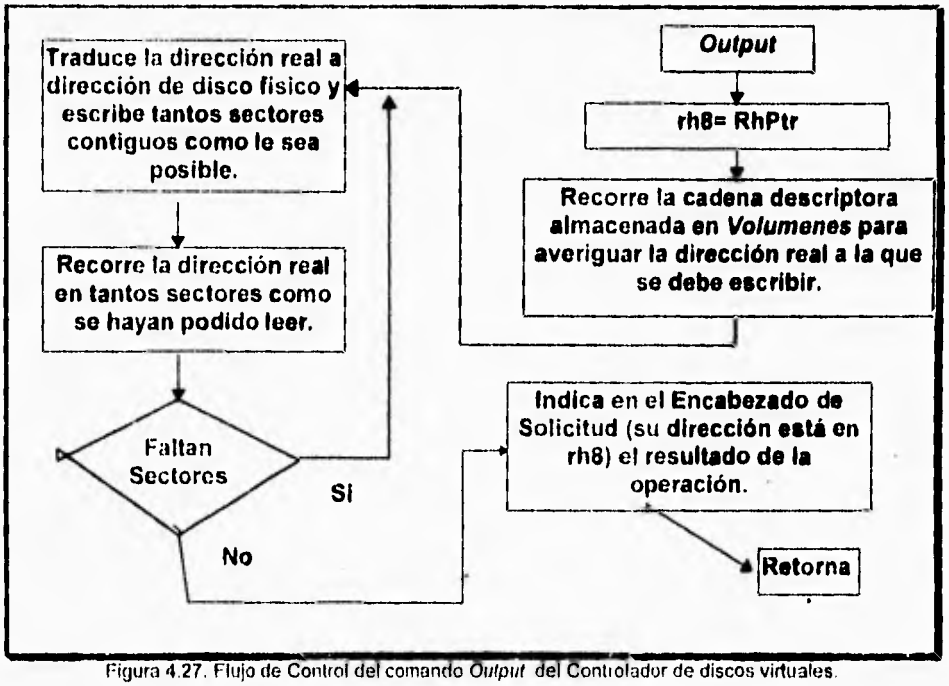



Figura 4.27. Flujo de Control del comando *Output* del Controlador de discos virtuales.

**El siguiente diagrama de flujo de información (figura 4.14) muestra las estructuras de datos que este comando utiliza:**

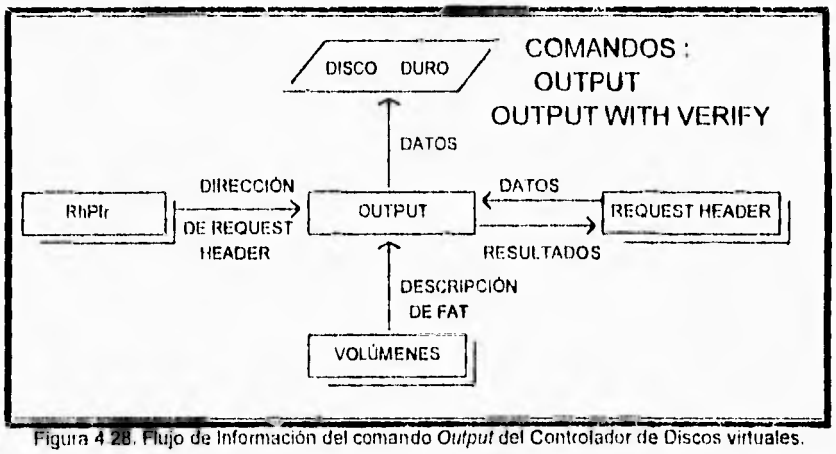



Figura 4.28. Flujo de Información del comando *Output* del Controlador de Discos virtuales.

Durante la descripción de las funciones de cada comando que se acaba de realizar se hizo mención de diferentes entidades utilizadas como fuente o destino de información. A continuación se definirá cada una de ellas. Durante la exploración del código fuente se verá que aparecen otras, sin embargo se consideró mostrar solo las de mayor importancia y dificultad de comprensión.

## Entidades relacionadas y su representación.

### rhptr.

Es una variable de tipo Apuntador de 4 bytes cuyo objetivo es almacenar la dirección del **Request header** (2 bytes usados para segmento, 2 bytes para offset, como una variable tipo Far Pointer en lenguaje C). En cada llamada a un comando de Entrada/Salida su valor es modificado por la rutina **ESTRATEGIA** y consultado por la rutina **INTERRUPCIÓN** (para averiguar el número de comando e invocarlo) y por las rutinas de atención a comandos (para recibir o enviar parámetros a DOS a través del **Request header**). Su sitio de definición es el código base en ensamblador y se referencia desde el código C, en el cual se declara como variable externa. Su descripción es la siguiente:

Desde el código base en ensamblador:

```
public _rhptr
_rhptr      equ   $
rh_ofs      dw    ?
rh_seg            dw    ?
```

Desde el código C:

```
extern rh_t   far * rhptr ;
```

donda **rh_t** es una estructura que define al Encabezado de Solicitud.

### File_System.

Es un conjunto de variables (que en lenguaje C se decidió representar como una estructura) cuyo objetivo es describir el sistema de archivos (file system) sobre el que está trabajando el Controlador. Su valor es establecido durante la rutina de atención al comando **INICIALIZATION** y después ya nunca cambia. Solo es usado por las rutinas que realizan lecturas o escrituras de bajo nivel sobre el disco duro. Las rutinas de nivel alto no hacen referencia a el. Las variables que

comprende describen diferentes parámetros de importancia para la descripción física y lógica de un disco duro**:

- Valor físico de la unidad de disco duro en que se trabaje (128=primer disco físico, 129=segundo disco físico, etcétera).
- Número de la partición sobre la que se trabaja.
- Sector de inicio y sector base del Registro de Arranque de la partición en que se trabaja.
- Tipo de Tabla de Registro de Archivos (FAT de 12 o 16 bits).
- Cantidad de sectores por cluster.
- Número de sectores reservados en el file system.
- Número de Tablas de Registro de Archivos (FAT's) y de archivos en el directorio raiz.
- Número de sectores que ocupa cada Tabla de Registro de Archivos (FAT) del Sistema de archivos.
- Número de sectores ocultos (Los que hay antes del inicio de la partición).
- Las direcciones de inicio de la primer FAT, el directorio raiz y el área de datos, relativas al inicio del Sistema de archivos en que se trabaje.
- Dirección del sector de FAT que actualmente se tenga en el buffer de trabajo, también relativa.
- Los tamaños de cada FAT, del directorio raiz y del volúmen completo, medidos en sectores.
- Un buffer del tamaño de un sector (512 bytes) para realizar lecturas de FAT.

La representación de la variable file system en lenguaje C es la siguiente:

```
struct file_system {
  BYTE      drive;              /* Unidad del File System */
  BYTE      particion;          /* Partición del File System*/
  DWORD     inicio;             /* # de Sector del inicio de Partición */
  DWORD     base;               /* Sector base del Boot*/
  BYTE      bpFAT;              /* # de Bits Por Celda de FAT */
  BYTE      spCl;               /* Sectores Por Cluster */
  WORD      res_sectores;       /* Numero de Sectores Reservados */
  BYTE      num_FATs;           /* # de FATs          */
  WORD      archs_raiz;    /* # de archivos en Dir. Raiz */
  WORD      spFAT;         /* # de Sectores Por FAT   */
  DWORD     ocultos;       /* # de Sectores ocultos   */
```

** Para comprender el significado de algunos de los parámetros que la estructura de datos File_System define es necesario conocer al menos superficialmente la estructura física de un disco duro, así como la definición de un File System (Sistema de Archivos) bajo el Sistema Operativo DOS. El proporcionar una referencia detallada de estos aspectos está más allá del alcance de este trabajo. Sin embargo, se provee un apéndice que cubre de manera clara y resumida los conceptos necesarios para comprender los conceptos presentados en la construcción del Controlador.

```
  WORD    eFATps;              /* # de celdas de FAT Por Sector   */
  DWORD   FAT_rba;       /* RBA del primer sector de FAT */
  DWORD   DIR_rba;       /* RBA del primer sector de Directorio */
  DWORD   DATA_rba;            /* RBA del primer sector de datos   */

  DWORD   FAT_cargada;         /* RBA de FAT actualmente en Buffer */

  WORD    tamano_FAT;          /* # de Sectores por FAT   */
  WORD    tamano_DIR;          /* # de Sectores por DIR   */
  DWORD   tamano_VOL;          /* # de Sectores por Volumen */

  BYTE    *FAT_ptr;      /* Apuntador al buffer de FAT  */
};
```

## Volúmenes.

**Conjunto de variables utilizado para soportar las características del o de los discos virtuales con que se esté trabajando. Su valor es establecido durante la rutina de Inicialización, de acuerdo a la línea que se haya pasado como parámetro al controlador durante su inicialización. Es un vector de 5 elementos, donde cada uno contiene la siguiente información:**

- **Nombre del archivo que representa al disco virtual, y que siempre se ubica en el directorio raíz del file system en que se trabaje.**
- **Posición del archivo en al directorio raíz.**
- **Tamaño total del archivo que representa al disco virtual, en sectores.**
- **Un vector de máximo 40 elementos, donde cada uno indica una posición dentro de la FAT y una cantidad de sectores continuos a partir de dicha posición. Refleja la distribución del archivo a través del disco duro, por medio de la cadena de FAT. Se puede considerar aumentar el valor de 40 si la fragmentación del disco duro es excesiva.**

**Se representa en lenguaje C de la siguiente forma:**

```
/*
Estructura para mantener información acerca de la posición del
Archivo-Disco Virtual en la FAT...
*/

struct Cadena_FAT {
      WORD Cluster ;
      WORD Cuantos ;
};
```

```
  WORD    eFATps;              /* # de celdas de FAT Por Sector  */
  DWORD   FAT_rba;       /* RBA del primer sector de FAT */
  DWORD   DIR_rba;       /* RBA del primer sector de Directorio */
  DWORD   DATA_rba;            /* RBA del primer sector de datos  */

  DWORD   FAT_cargada;         /* RBA de FAT actualmente en Buffer */

  WORD    tamano_FAT;          /* # de Sectores por FAT  */
  WORD    tamano_DIR;          /* # de Sectores por DIR  */
  DWORD   tamano_VOL;          /* # de Sectores por Volumen */

  BYTE    *FAT_ptr;      /* Apuntador al buffer de FAT  */
};
```

## Volúmenes.

Conjunto de variables utilizado para soportar las características del o de los discos virtuales con que se esté trabajando. Su valor es establecido durante la rutina de inicialización, de acuerdo a la línea que se haya pasado como parámetro al controlador durante su inicialización. Es un vector de 5 elementos, donde cada uno contiene la siguiente información:

- Nombre del archivo que representa al disco virtual, y que siempre se ubica en el directorio raíz del file system en que se trabaje.
- Posición del archivo en el directorio raíz.
- Tamaño total del archivo que representa al disco virtual, en sectores.
- Un vector de máximo 40 elementos, donde cada uno indica una posición dentro de la FAT y una cantidad de sectores continuos a partir de dicha posición. Refleja la distribución del archivo a través del disco duro, por medio de la cadena de FAT. Se puede considerar aumentar el valor de 40 si la fragmentación del disco duro es excesiva.

Se representa en lenguaje C de la siguiente forma:

```
/*
Estructura para mantener informeción acerca de la posición del
Archivo-Disco Virtual en la FAT...
*/

struct Cadena_FAT {
      WORD Cluster ;
      WORD Cuantos ;
};
```

```
/*
Estructura para mantener información acerca del
Archivo-Disco Virtual en la FAT...
*/
typedef struct D_V {
        BYTE Nombre [14] ;
        WORD Pos_Dir ;
        WORD Tot_Sects ;
        struct Cadena_FAT  Cadena_Virtual [40] ;
} Disco_Virtual ;

/*
Vector de Volúmenes...
*/
Disco_Virtual   Volumenes [5] ;
```

## Request Header.

Es la estructura definida por DOS para comunicar información en ambos sentidos entre el Sistema operativo y un controlador de dispositivos. Tiene una parte fija (La cual es invariante para todos los comandos) y una parte variable dependiendo del comando que se ejecute. Su formato está preestablecido, por tanto solo se mostrará la forma en que se implementa en lenguaje C para cada comando:

```
/*
        Sección fija del Request Header...
*/
typedef struct rhfixed_struct {
        BYTE   longitud ;
        BYTE   unidad ;
        BYTE   comando ;
        WORD   status ;
        BYTE   res [8] ;
} rh_t ;

/*
        Secciones variables del Request Header...
*/
typedef struct rh_init_struct {
        rh_t          rh ;
```

```
    BYTE        nunidades ;
    void        (far * dir_break) (void);
    bpb_t       * far * bpbtab ;
    char        drive ;
} rh0_t ;

typedef struct rh_media_check_struct {
    rh_t        rh ;
    BYTE        media ;
    BYTE        md_stat ;
    char far    * volid ;
} rh1_t ;

typedef struct rh_get_bpb_struct {
    rh_t        rh ;
    BYTE        media ;
    char far    * buf ;
    bpb_t far   * bpb ;
} rh2_t ;

typedef struct rh_ioctl_struct {
    rh_t        rh ;
    BYTE        media ;
    char far    * buf ;
    WORD        cantidad ;
    WORD        inicio ;
} rh3_t, rh12_t ;

typedef struct rh_io_struct {
    rh_t        rh ;
    BYTE        media ;
    char far    * buf ;
    WORD        cantidad ;
    WORD        inicio ;
    char far    * volid ;
} rh4_t, rh8_t, rh9_t ;
```

**Como se puede ver, más de un comando puede usar un mismo tipo de parte variante.**

# Reporte de la Implementación.

## Herramientas a usar.

La selección de herramientas no representa demasiada dificultad, pues no se requiere de software ni de hardware costoso o de difícil adquisición.

Específicamente hablando, las herramientas utilizadas durante las pruebas y para el desarrollo final fueron:

1. Una computadora PC-80386 con un disco duro y MS-DOS versión 6.2.
2. Debug, Herramienta contenida en el sistema operativo DOS.
3. ProView, Herramienta de monitoreo de memoria de McAfee Associates.
4. Q, Editor ASCII multi-archivos común y corriente.
5. MASM, Macroensamblador de Microsoft.
6. TCC, Compilador en línea de Turbo C++ de Borland.
7. TLINK, Preparador en línea incluido con los productos Borland.
8. TLIB, Programa manejador de librerías de código de Borland.
9. ARRANGE, programa de utilería de propia creación.
10. MAKE, utilería de "C" para integrar las herramientas de desarrollo.

## Mecánica de Construcción.

De la breve explicación acerca de la estructura física externa de un Controlador de Dispositivos mencionada en el capítulo sobre controladores en DOS se infiere que un Controlador de Dispositivos es un programa de una estructura rígida y completamente bien delimitada. En la aplicación práctica desarrollada para este trabajo la metodología a seguir para lograr dicha estructura fue la siguiente:

1. La estructura base del Controlador (la que proporciona la mayor parte de esta estructura rígida y bien delimitada) se programó en ensamblador, para resolver problemas como el ordenamiento de secciones requerido por la definición de estructura de un Controlador de Dispositivos.
2. Para mayor facilidad y velocidad de desarrollo, los comandos a que el dispositivo debía responder fueron programados, depurados y probados separadamente en lenguaje C.
3. Los conflictos de referencias a variables o etiquetas externas (entre los códigos Ensamblador y C) se resolvieron.
4. Usando MASM (Ensamblador de Microsoft) y Turbo C (compilador de C de Borland) se compilaron los códigos fuente.
5. El código resultante del lenguaje C es código fuente ensamblador, sobre el cual se corrió un programa utilería de propia ceación llamado ARRANGE el cual fue útil para resolver de manera automática los conflictos de

**ordenamiento de llamadas debidos al compilador de lenguaje C, y después se generó el código objeto correspondiente.**

6. **El código objeto resultante del paso anterior se introdujo en una librería de código usando TLIB (Manejador de librerías que acompaña a Turbo C).**
7. **Usando TLINK (ligador de código que acompaña a Turbo C) se produjo el código en formato .COM final, proporcionándole como entradas el código objeto proveniente del ensamblador, la librería estándar de "C" y la creada en el paso anterior.**

**El diagrama siguiente ilustra el proceso descrito:**

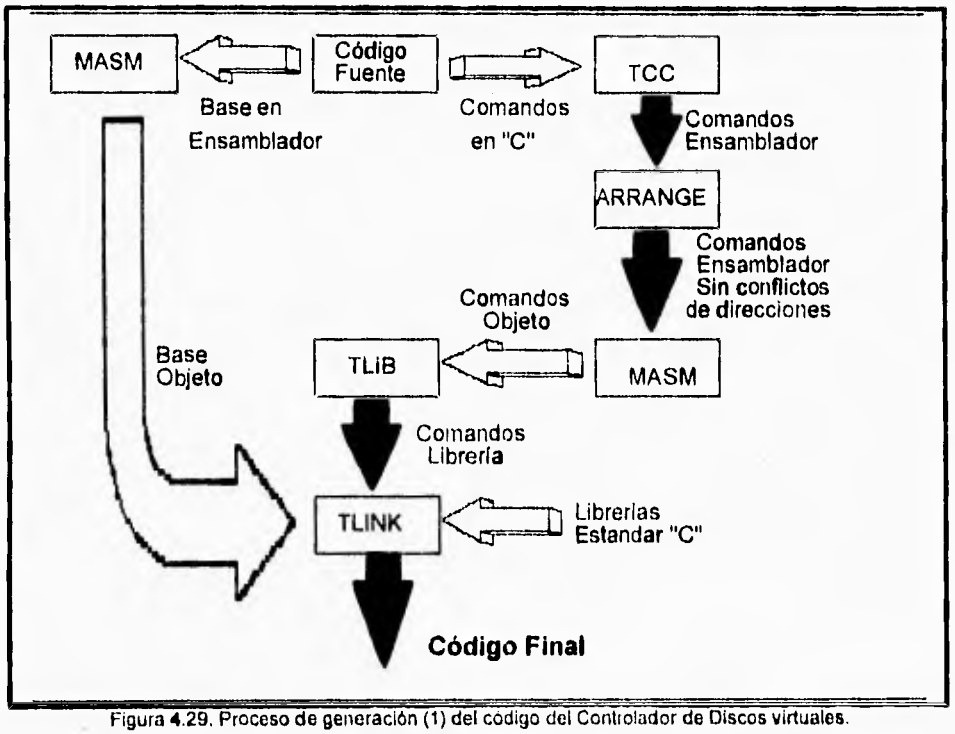



Figura 4.29. Proceso de generación (1) del código del Controlador de Discos virtuales.

**Es importante resaltar que, si bien el proceso descrito parece complicado, es solo el resultado de una buena cantidad de pruebas en las que mostró ser el que ofrecía un mejor balance en cuanto a sus características de velocidad y sencillez de comprensión y de creación. Además, se automatiza completamente gracias al uso de la utilería Make que el compilador de Turbo C incluye. Analizando el esquema presentado se puede argumentar que resulta muy difícil usar las**

herramientas tradicionales de depuración de manera efectiva sobre el sistema, pero ésta es una debilidad que todo Controlador de dispositivos padece durante su desarrollo debido a su particularidad de trabajar siempre a nivel de BIOS.

Para terminar esta sección, se incluye en la figura 4.16 el flujo de control seguido para crear el archivo final:

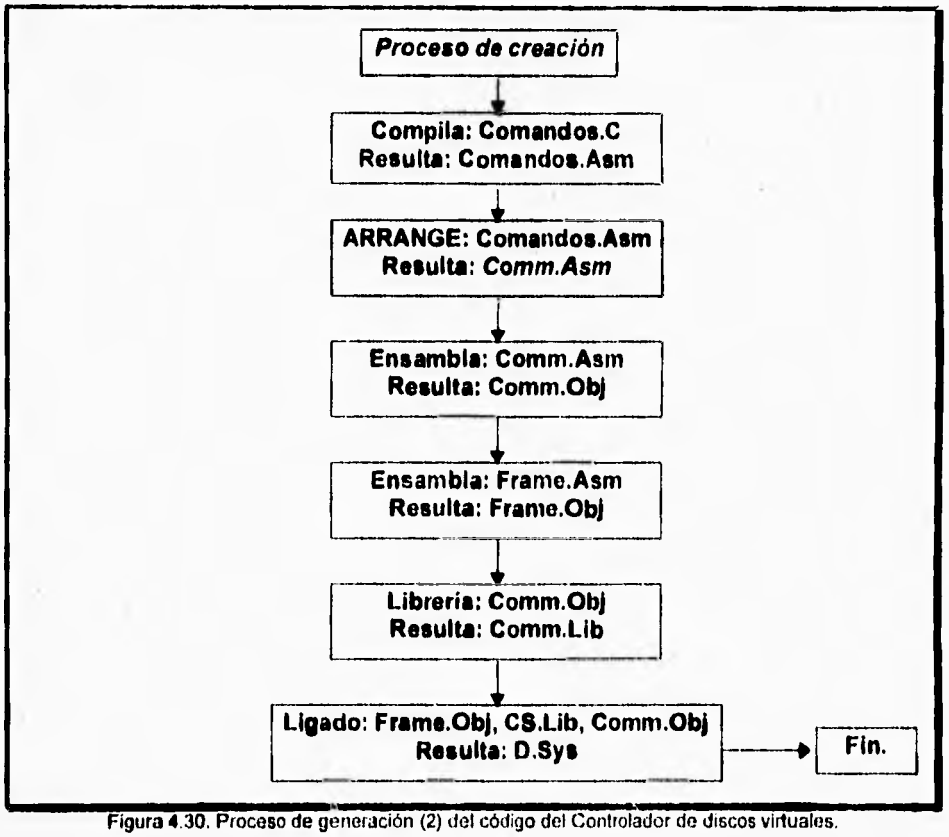



Figura 4.30. Proceso de generación (2) del código del Controlador de discos virtuales.

## Comentarios técnicos acerca de la implementación.

Esto capítulo está dedicado a clarificar algunos de los aspectos referentes a la codificación del proyecto. No se pretende explicar cada detalle de la

implementación sino solo hacer enfasis en aquello que parezca obscuro o que haya planteado un problema cuando el código se escribió.
Por simplicidad el código fuente no se colocó en esta parte del texto, sino en uno de los apéndices al final del documento.

## Conflictos de direccionamiento entre ensamblador y C.

Al inicio de la sección dedicada al diseño del controlador se indicó el porqué de la selección de ensamblador y C para la implementación. Entre las conclusiones se indicó la decisión del uso de Lenguaje Ensamblador para programar la plataforma del controlador gracias a que proporciona control total sobre la estructura del código (impuesta por la definición de un Controlador en DOS), y el uso de lenguaje C para implementar el código de los comandos pues proporciona mayor facilidad de programación de código de cierta complejidad. Esta combinación, en nuestra opinión, es la más adecuada para el proyecto. Sin embargo, este acercamiento no es perfecto. Su mayor defecto está en su necesidad de conciliar el código ensamblador con las convenciones de nombramiento de grupos que el compilador de lenguaje C realiza al generar su código. Afortunadamente, este problema pudo resolverse sin demasiada dificultad después de las complicaciones que causó inicialmente. El problema es el siguiente:

El compilador de C (para compiladores C++ puede diferir) divide de la siguiente forma un programa modelo tiny:

- _DATA: Área dedicada al almacenamiento de variables globales no inicializadas.
- _BSS: Área dedicada al almacenamiento de variables globales inicializadas.
- CONST: Área dedicada al almacenamiento de constantes.
- _TEXT: Área dedicada a almacenar código.

Debido a que ambos ambientes comparten datos y código, todo lo que tendria que hacerse antes de ligar ambos es definir estas mismas áreas en el código ensamblador y guardar en cada una de ellas lo que corresponda (datos o código).
Sin embargo, existe otro problema: El compilador de C, para simplificar su esquema de direccionamiento de variables, define un grupo llamado DGROUP en el cual incluye a las áreas _DATA y _BSS. El área _TEXT es dejada fuera de este DGROUP. Respecto a los segmentos de código y datos, el primero queda indicando al área _TEXT y el segundo al grupo DGROUP. El problema con esta situación es que el código objeto del controlador necesita estar referenciado a un solo inicio, común para datos y código.
Esta dificultad se resolvió anexando el área _TEXT al grupo DGROUP y haciendo que el segmento de código inicie también en DGROUP (esto último es redundante pero clarifica la estructura del programa). A continuación se muestra parte del código ensamblador generado por el compilador de C, antes y después de que las modificaciones realizadas:

**Antes de la modificación:**

```
DGROUP      group _DATA,_BSS
assume      cs:_TEXT,ds:DGROUP
```

**Después de la modificación:**

```
DGROUP      group _DATA,_BSS,_TEXT
assume      cs:DGROUP,ds:DGROUP
```

Se reemplazan además todas las referencias al área _TEXT que aparezcan a lo largo del código por DGROUP.

En el recuadro inferior se muestran las modificaciones hechas al código ensamblador:

```
; Definición del grupo:

DGROUP  group _DATA, CONST, _BSS, _TEXT

_DATA   segment word public 'DATA'
     assume ds:DGROUP
               ; código
_DATA ends

CONST   segment word public 'CONST'
     assume  ds:DGROUP
               ; código
CONST   ends

_BSS    segment word public 'BSS'
     assume ds:DGROUP
               ; código
_BSS    ends

_TEXT   segment word public 'CODE'
     assume  cs:DGROUP, ds:DGROUP
               ; código
_TEXT   ends
```

## Uso de la utilería Arrange.

En el apartado anterior se indicó la forma de resolver los conflictos de direccionamiento que se presentan entre los lenguajes ensamblador y C. El método que se presentó es sencillo y fácil de implementar. Sin embargo, tiene un problema: Para modificar las instrucciones que se mostraron para el código en lenguaje C, se requiere el código equivalente en lenguaje ensamblador. Éste se genera usando la configuración de compilador adecuada y después sobre este código se realizan los cambios. El problema es que ésto se debe realizar cada vez que se modifique el código en lenguaje C aún si los cambios son mínimos; el efecto consiguiente es una obstaculización en la rapidez del desarrollo. La solución a ésto fue la implementación de un programa en lenguaje C, ARRANGE, el cual trabaja sobre el código ensamblador generado por el compilador de C realizando los cambios antes indicados. Para lograr mayor generalidad, ARRANGE trabaja a partir de las instrucciones almacenadas en un archivo de configuración tipo ASCII semejante al siguiente:

```
s/DGROUP   group _DATA,_BSS/DGROUP     group _DATA,_BSS,_TEXT/
s/assume           cs:_TEXT/assume     cs:DGROUP/
```

Cada línea se interpreta de la siguiente forma: La letra "s" indica que se inicia una definición de substitución. A partir de la primera diagonal y hasta la segunda se encuentra el texto que el programa deberá buscar, y entre la segunda y la última el texto substituto. Esta utileria no es de propio desarrollo. Fue modificada a partir del código encontrado en uno de los libros citados como fuente en la bibliofrafía. Sin embargo su uso agilizó el desarrollo significativamente.

## Cálculo de la dirección de Fin de código.

A lo largo del código fuente del controlador se encuentran secciones cuyo significado puede parecer obscuro. Todas éstas pueden después de un poco de observación comprenderse fácilmente. Sin embargo algunas de éstas son un tanto más difíciles de entender. Una de ellas en nuestra opinión requiere un poco de atención extra. Ésta se da debido a lo siguiente: En la rutina de atención al comando de Inicialización se puede retornar a DOS una dirección dentro del código objeto del controlador a partir de la cual DOS "corte" el código y así ahorre memoria RAM. La sección del controlador que de esta forma se elimina contiene por lo común rutinas o variables que se usaron solo durante el proceso de inicialización y que ya no volverán a ser útiles durante el resto de la operación del controlador. Por este motivo, Tradicionalmente estas rutinas son las que se encuentran en la parte final del archivo que contiene al controlador. En este caso en particular, las instrucciones que se utilizaron para calcular el punto de corte son las siguientes:

```
/* Variable definida globalmente */
extern void     estrategia (void) ;

/* Variable definida dentro de la rutina de inicialización */
rh0_t far * rh0 ;

 /* Indica la Dirección del fin de código... */
 rh0 -> dir_break = (void (far *) (void)) inicializa ;
 (unsigned int) rh0 -> dir_break += (unsigned int) estrategia ;
```

**En el recuadro, La variable definida como estrategia es un apuntador a la rutina del mismo nombre y que, como ya se vio en el capítulo acerca de controladores de dispositivos en DOS, es siempre invocada antes de llamar a la rutina de atención de cada comando. En nuestro controlador el cuerpo de esta rutina es el primero del área donde el código se encuentra.**
**La variable local rh0 es un apuntador al Request Header que se usa para comunicar a DOS con el controlador. En la variable dir_break de este Request Header se deposita la dirección de rompimiento. La siguiente figura ilustra la dirección de los elementos involucrados en este cálculo dentro del controlador:**

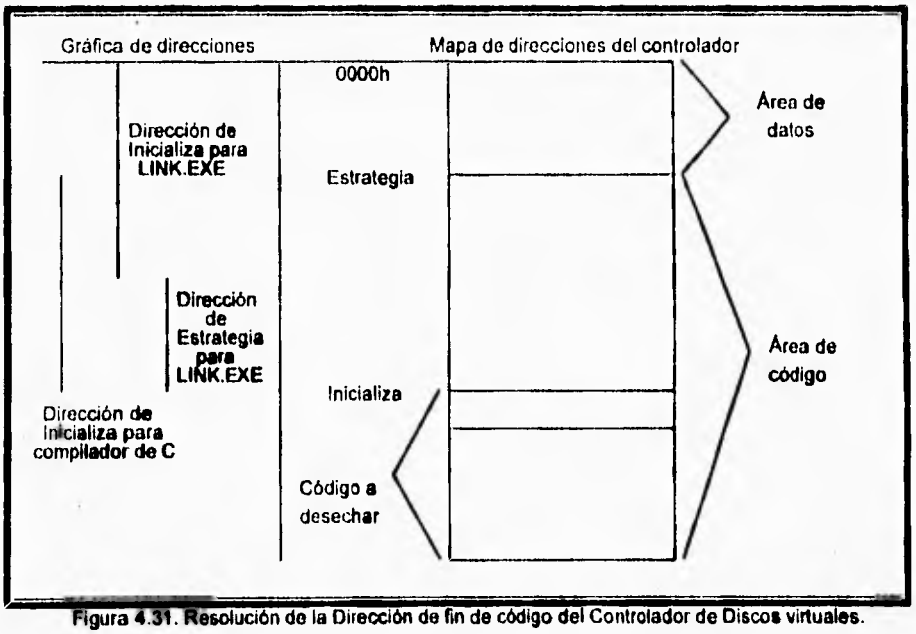



Figura 4.31. Resolución de la Dirección de fin de código del Controlador de Discos virtuales.

**Notese en la ilustración anterior que incluso el código del comando inicializa se desecha. A continuación se explicará como se llega a la dirección final.**
**La primera instrucción del código relacionado es:**

```
rh0 -> dir_break = (void (far *) (void)) inicializa ;
```

**En ella se asigna a la variable destino el valor del apuntador al inicio de la función inicializa. Aparentemente, ésto sería suficiente para indicar a DOS que queremos que elimine al código a partir de la función inicializa. Sin embargo, no olvidemos un detalle: El compilador de C presupone que el código que el genera está antes que el área de datos. Por tanto, calcula la dirección de la función de inicialización partiendo desde Cero. El programa TLINK.EXE, por su parte, ha sido instruido a colocar el área de datos antes que la de código. Ésto implica que el valor que el compilador le proporcionó como dirección inicial de la rutina de Inicialización está en realidad referenciando un área de memoria bastante menor. Ésto hace necesario sumar a dicho valor tantas unidades como bytes tenga el área de datos. Como se advierte en la ilustración anterior, la rutinaEstrategia está al final del área de datos y por lo tanto es idónea para resolver este problema. La siguiente instrucción lleva ésto a cabo:**

```
(unsigned int) rh0 -> dir_break += (unsigned int) estrategia ;
```

Podría suponerse que la variable estrategia tiene un valor Cero para TLINK.EXE por las mismas razones que la variable inicializa es incorrecta. Sin embargo, la variable estrategia está definida como Externa y por tanto su valor no ha sido definido al momento de compilación. Su valor se resuelve hasta que se llega a TLINK.EXE y por tanto es real.

## Inclusión de una pila de datos local.

**El formato de los archivos que contienen programas Controladores de Dispositivos es semejante al de los archivos ejecutables tipo .COM, y el código que contienen sigue las convenciones del modelo de memoria tiny (solo 64 KiloBytes para los segmentos de datos, pila y código). La pila (stack) crece de las áreas altas de memoria a las bajas. Si el código hace un uso extensivo de la pila, si hay un área considerable dedicada a datos o si el código es muy grande (o cualquier combinación de estos factores) el área de pila puede llegar a encimarse a la de código causando efectos impredecibles; la probabilidad de que se presente este problema es mayor si se considera la inclusión de código escrito en lenguajes de alto nivel pues éstos hacen un uso intensivo de pila (al hacer llamados a subrutinas). El código del controlador escrito en este trabajo no denota riesgos en este sentido. Sin embargo, se consideró prudente reservar un área fija de memoria dentro del propio controlador para dedicarla a usos de pila. Cada vez que un comando diferente es enviado por DOS al controlador se salva el**

entorno original (el valor de los registros al momento de entrar) y se cambia la pila que en ese momento exista por la interna. El código para realizar ésto es:

**Definición de las variables requeridas:**

```
stack_ptr     dw     ?                        ; Stack pointer original
stack_seg     dw     ?                        ; Stack segment original

newstack      db     100h dup('*')      ; Espacio para el nuevo Stack
newstacktop   label  word                     ; Marcador de inicio de Stack
dummy         db     ?                        ; Primer byte del Stack
```

**Para cambiar al Stack interno:**

```
    ; Realiza el cambio a la pila local...
    cli
    mov    cs:stack_ptr,  sp
    mov    cs:stack_seg,  ss
    mov    ax,            cs
    mov    ss,            ax
    mov    sp,            newstacktop
    sti
```

Para retornar a la pila original

```
    ; Retorna a la pila original
    cli
    mov    ss,    cs:stack_seg
    mov    sp,    cs:stack_ptr
    sti
```

## Archivo MAKE usado para la creación del código final.

Para automatizar el proceso de generación del código final se usó la utilería MAKE, incluida con la mayoría de los compiladores de lenguaje C disponibles. La entrada a esta utilería es un archivo ASCII que indica las dependencias de los archivos fuente involucrados, y la secuencia de pasos requerida para crear el código. En nuestro proyecto dicho archivo fue llamado **MAKEFILE**, y refleja los pasos mostrados en la sección acerca de la mecánica de construcción de nuestro controlador, donde además se incluyó un diagrama de flujo (atrás en este mismo capítulo). A continuación se muestra el contenido de **MAKEFILE**:

```
#
# MakeFile para el device driver con interfaz Ensamblador - lenguaje C.
#
#       Banderas para compilación:
#       -S    : Genera código ensamblador
#       -C    : Permite comentarios anidados
#       -c    : Compila solamente
#       -mt   : Modelo Tiny
#       -N-   : deshabilita chequeo de stack overflow
#       -v-   : Deshabilita depuración a nivel Fuente
#       -P-   : Deshabilita generación de código C++
#       -a-   : Compactación de estructuras
#       -I\develop\tc\include  : Ruta de archivos incluidos

FLAGS_COMP  =  -M -S -C -c -mt -N- -v- -P- -a- -Id:\develop\tc\include

d.sys:          frame.obj     comm.lib
                tlink /t /m /s frame.obj , d.sys, d.map, cs.lib comm.lib

comm.lib:       comm.obj
                tlib comm.lib /e +-comm.obj

frame.obj:      frame.asm
                masm    frame.asm;

comm.obj:       com.arr comandos.asm
                arrange com.arr  comandos.asm comm.asm
                masm comm.asm ;


comandos.asm: comandos.c frame.h initdefs.h accdisco.h utileria.h varios.h
                                                        output.h input.h init.h
                tcc   $(FLAGS_COMP) comandos.c
```

La forma en que se interpreta el archivo es la siguiente:

1. El archivo se comienza a leer de abajo hacia arriba.
2. El archivo se puede dividir en grupos, los cuales son visibles en el archivo.
3. En cada grupo, sobre la columna izquierda se indica el nombre del archivo que se quiere crear, y de lado derecho en el primer renglón se indican los archivos de los que el archivo nombrado a la izquierda depende. Es decir, cuando cualquiera de los archivos nombrados a la derecha se modifique entonces el archivo nombrado a la derecha se volverá a crear. En el renglón o

**renglones inferiores de cada grupo se encuentran los comandos que se deberán ejecutar cada vez que el archivo mencionado a la derecha se cree.**

**Basandonos en ésto, vemos que el archivo comandos.asm (que contiene el código ensamblador generado por el compilador de C para los comandos) se crea ejecutando el programa tcc (compilador en linea de Turbo C).**
**En el siguiente grupo se crea el programa objeto comm.obj, primero resolviendo los conflictos de direcciones de comandos.asm con la utileria ARRANGE (descrita más arriba en este capitulo), y después ensamblando con MASM el código resultante.**
**A continuación se obtiene el código objeto de frame.asm (el cual contiene el código base del ensamblador) ensamblando éste último con MASM.**
**El penúltimo grupo obtiene el archivo de libreria comm.lib a partir de la ejecución del programa TLIB (manejador de librerias de código de Turbo C) con comm.obj como entrada. El propósito de introducir comm.obj en la libreria fue el de poder dejar al final el código del comando inicializa dentro del código final, y asi poder reducir el código del controlador para ahorrar memoria (esta caracteristica ya fue descrita en una parte de esta sección).**
**Finalmente el grupo restante crea el código final ligando frame.obj (la base del controlador), código dentro de cs.lib (libreria de rutinas estándar de C para el modelo tiny) y el código dentro de comm.lib (la libreria que contiene el código de respuesta a los comandos). Vease que esta libreria se dejó al final para poder hacer la reducción de código durante la inicialización.**

# 5. Conclusiones.

**Conclusiones.**

# Conclusiones.

Es esta sección se presentan las conclusiones a las que se llegó con el desarrollo de la aplicación propuesta para este trabajo.

La aplicación se desarrolló conforme a las especificaciones dictadas en las secciones de Análisis y Diseño del capítulo 4. Revisando los objetivos por renglón se obtuvo lo siguiente:

1. Fácil acceso físico a la información.
- El Controlador de discos virtuales se puede instalar en un computadora PC con las librerías en disco duro. Las librerías pueden ser accesadas con la computadora PC trabajando independiente o ser accesadas vía Red mediante utilerías especiales (Al menos en el caso de SeCoFI, donde la red instalada es *Novell Netware 3.11*, la cual no soporta compartición de recursos de las Estaciones de trabajo. En este caso se utilizó la utilería llamada*MapAssist*); A este respecto, faltó realizar pruebas con Ambientes operativos de red de arquitectura Compañero a Compañero, como *Windows para Grupos de trabajo* o *Lantastic*. Sin embargo, se debe mencionar que las restricciones que por facilidad de desarrollo se impusieron de que las librerías se localizaran solo en la raíz del primer disco duro de la computadora hicieron menos práctica a la aplicación.
2. Facilidad y Transparencia en el manejo.
- Debido a que la librerías se manejan a través de una Unidad lógica del sistema como si fueran discos comúnes, cualquier usuario con conocimientos promedio del Sistema operativo DOS puede utilizarlas.
3. Flexibilidad en la Creación, Recuperación y Actualización de información.
- La utilería proporcionada (*CreaDisk*) permite crear librerías de manera fácil y rápida. El método de acceso a la Librería permite de manera inherente el acceso a piezas separadas de software o a grupos, gracias a la estructura jerárquica que se puede representar en un disco DOS.
5. Rapidez, Eficiencia, Economía.
- La técnica de uso de unidades de disco utilizada por DOS es de acceso aleatorio, lo cual agiliza los accesos a información específica dentro de la librería.

Como toda aplicación práctica, el Controlador de discos virtuales es suceptible de ser modificado para obtener mejoras o capacidades adicionales. Algunas de los cambios que se podrían hacer a este trabajo en un futuro son:

- **Se requiere generalizar los algoritmos de búsqueda de archivos que utiliza el Controlador para localizar sus librerías. Actualmente solo localiza archivos en el directorio raiz del primer disco de la computadora. Para solucionar ésto se podrían utilizar algoritmos recursivos que buscaran archivos en subdirectorios.**
- **La utilería *CreaDisk* crea un archivo con el formato necesario para ser considerado como disco virtual. Para hacerlo pide como parámetro el tamaño del disco y crea un archivo de longitud equivalente. Ésto implica que el tamaño de la librería es siempre fijo, lo cual puede considerarse como un desperdicio de espacio útil. Una mejora significativa podría consistir en la modificación del Controlador y de la utilería *CreaDisk* para trabajar con librerías de tamaño variable, las cuales crecieran conforme los requerimientos de espacio dentro de la librería aumentaran. Debe advertir, sin embargo, que lograr ésto no es trivial.**
- **El Controlador de discos virtuales, debido a las restricciones de uso solo de Servicios BIOS, trabaja únicamente sobre discos duros en la computadora local. Sin embargo, se pueden realizar desarrollos semejantes con capacidades similares trabajando sobre ambientes de red, con Lectura/Escritura sobre puertos Seriales o Paralelos, etcétera.**
- **Se pueden agregar capacidades al Controlador mediante el paso de parámetros extra en la linea de configuración (en el archivo Config.Sys) o programando los comandos IOCTL y escribiendo una utilería que se comunique con el Controlador mediante esta vía. Se lograría asi que el Controlador cambiara su conducta en tiempo real. Algunas de las capacidades que se podrían agreagr al Controlsdor mediante esta vía son:**
  - **Despliegado de código de actividad en pantalla, con el fin de depurar o solo ilustrativamente.**
  - **Mantenimiento y obtención de cifras estadísticas como cantidad o frecuencia de accesos, comandos ejecutados, etcétera.**
  - **Implementación de niveles de Seguridad internos, mediante los cuales solo se pueda accesar la librería usando Contraseñas de entrada.**
  - **Realizar intercambio dinámico de librerías. Es decir, cambiar la asignación de Unidad lógica a otro archivo de librería sin tener que reconfigurar y arrancar la máquina.**

**Como se puede ver, El campo de los Controladores de dispositivos es sumamente amplio y lleno de posibilidades, solo limitado por la imaginación. El haber realizado este trabajo proporcionó al autor un conocimiento un poco más detallado acerca de los procesos intrínsecos de operación de una microcomputadora y constituyó una experiencia altamente provechosa.**

# 6. Apéndices, Glosario, Bibliografía.

## Apéndices.

A. Código fuente del controlador.

B. Código fuente de la utilería ARRANGE.

C. Código fuente de la utilería CREADISK.

D. Estructuras de un disco duro: A)Física genérica y B)Lógica en DOS.

E. Servicios BIOS de Disco duro.

F. Formato de la Interfaz a un Controlador de dispositivos.

## Glosario.

## Bibliografía.

# Lista de Tablas para el Capítulo 6.

# Lista de Figuras para el Capítulo 6.

# Apéndices

## Apéndice A. Código fuente del controlador.

**Este apéndice contiene el código fuente del controlador desarrollado, distribuido en diferentes archivos. La forma en que esta distribución se realizó es ilustrada a continuación :**

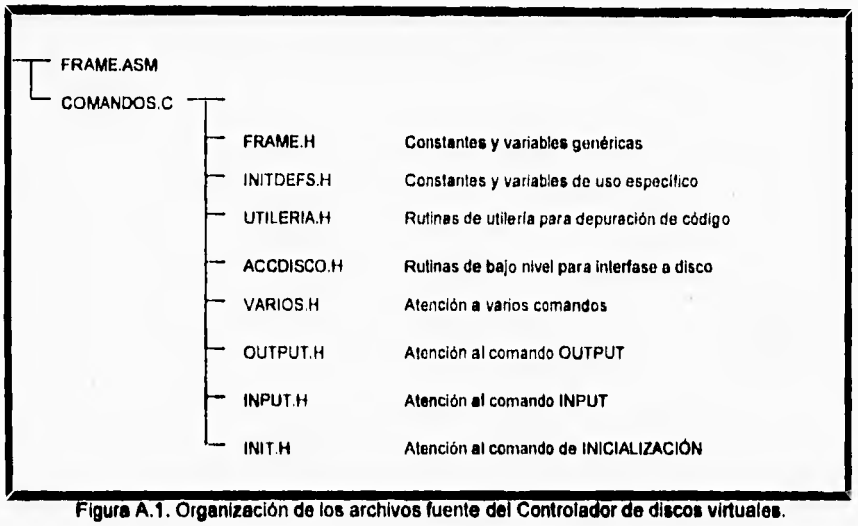



Figura A.1. Organización de los archivos fuente del Controlador de discos virtuales.

**Como se puede apreciar la distribución del código se divide en dos ramas:**

1. **La plataforma del Controlador escrita en lenguaje ensamblador (contenida en el archivo Frame.Asm), la cual lo estructura de acuerdo a los requerimientos de DOS. Contiene llamadas a rutinas externas de atención a Comandos de E/S.**
2. **Los archivos que contienen las rutinas en lenguaje C para atender a los comandos de Entrada/Salida. El archivo Comandos.C define variables globales usadas para mantener información importante acerca de la estructura Física y Lógica del disco duro, registro de volúmenes lógicos a representar y cadenas descriptoras de archivos en FAT, etcétera. Además, contiene las referencias a los archivos:**
   - **Frame.H**
   - **InitDefs.H**
   - **Utileria.H**
   - **AccDisco.H**
   - **Implementación de comandos: Init.H, Input.H, Output.H, Varios.H**

# Código fuente de : FRAME.ASM

```
        name    driver_Discos
        page    60, 132
        title   Base Genérica de Device Drivers para ligar con código C...


;*********************************************
;       Instrucciones para el ensamblador...                    *
;*********************************************

; Todo el código en un solo grupo...
DGROUP  group _DATA, CONST, _BSS, _TEXT


_DATA   segment word public 'DATA'
        assume ds:DGROUP

;*********************************************
;       Encabezado del Device Driver...                         *
;*********************************************
            public _Encabezado_device
            org    0
_Encabezado_device      label   word
sig_dev         dd      -1
atributo        dw      2000h
estrategia              dw      DGROUP : _estrategia
interrupcion            dw      DGROUP : _interrupcion
nombre                  db      5       ;5 discos máximo
                db      7 dup(?)

;*********************************************
;       Definición del Request Header...                        *
;*********************************************
            public _rhptr
_rhptr          equ     $  ; Para guardar la
rh_ofs          dw      ?  ; diracción del
rh_seg          dw      ?  ; Request Header...
```

```
;***********************************************
;     Definiciones para la nuava pila local...
;***********************************************

stack_ptr      dw     ?   ; Dirección da la
stack_seg      dw     ?   ; pila original

newstack       db   100h dup('*')  ; Nueva pila
newstacktop    label  word        ; de 256
dummy          db     ?           ; bytes...

_DATA ends

;;; Definido para conformar con código C...
CONST  segment word public 'CONST'
       assume ds:DGROUP
CONST  ends

;;; Definido para conformar con código C...
_BSS   segment word public 'BSS'
       assume ds:DGROUP
_BSS   ends

_TEXT  segment word public 'CODE'
       assume cs:DGROUP, ds:DGROUP

;***********************************************
;     Procedimiento Estrategia...
;***********************************************
            public _estrategia
_estrategia    proc   far
       mov     cs:rh_seg, es  ; Guarda el
       mov     cs:rh_ofs, bx  ; Request Header
       ret
_estrategia    endp
```

```
;*********************************************
;     Procedimiento Interrupción...
;*********************************************
          public _interrupcion
_interrupcion  proc   far
     ; Salva el estado de la máquina
     ; en la pila original...
     cld
     push ds
     push es
     push ax
     push bx
     push cx
     push dx
     push di
     push si

     ; Realiza el cambio a la pila local...
     cli
     mov    cs:stack_ptr,  sp
     mov    cs:stack_seg,  ss
     mov    ax,            cs
     mov    ss,            ax
     mov    sp,            newstacktop
     sti

     ; Recupera la Dirección del Request Header...
     mov    ax,    cs:rh_seg
     mov    es,    ax
     mov    bx,    cs:rh_ofs

     ; Obtiene la dirección de la rutina de
     ; atención respectiva y le cede el control...
     mov    al, es:[bx] + 2
     rol    al, 1
     lea    di, TABLA_COMANDOS
     mov    ah, 0
     add    di, ax

     call   word ptr [di]
```

```
        ; Retorna a la pila original
        cli
        mov     ss,     cs:stack_seg
        mov     sp,     cs:stack_ptr
        sti

        ; Restaura ambiente para retorno a DOS...
        pop     si
        pop     di
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     es
        pop     ds
        ret

        ; Declaración de rutinas externas en C
        ; para ejecución de comandos...
        EXTRN   _inicializa             :near
        EXTRN   _mediacheck             :near
        EXTRN   _getbpb                         :near
        EXTRN   _input                  :near
        EXTRN   _output                 :near
        EXTRN   _com_descon             :near
        EXTRN   _pasa_sin_uso                   :near

        ; TABLA_COMANDOS es la tabla con
        ;  direcciones de las rutinas con código
        ;  de atención a cada comando...
TABLA_COMANDOS  label   word
        ; 00-Inicialización...
        dw  DGROUP:_inicializa
        ; 01-MediaCHK...
        dw  DGROUP:_mediacheck
        ; 02-Construye BPB...
        dw  DGROUP:_getbpb
        ; 03-IOCTL READ ( No implementada )...
        dw  DGROUP:_com_descon
        ; 04-Entrada de datos...
        dw  DGROUP:_input
        ; 05-Lectura no destructiva (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 06-Status de Entrada (No impl)...
        dw  DGROUP:_pasa_sin_uso
```

```
        ; 07-Vacia Buffers de entr (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 08-Escritura...
        dw  DGROUP:_output
        ; 09- " y Verif....
        dw  DGROUP:_com_descon
        ; 10-Estatus de salida (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 11-Vacia Buffers de sal (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 12-IOCTL WRITE (No implementada)...
        dw  DGROUP:_com_descon
        ; 13-Abre Device  (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 14-Cierra Device (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 15-"Medio Removible? (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 16-Escribe hasta Ocupado (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 17-No Definido (No impl)...
        dw  DGROUP:_com_descon
        ; 18-No Definido (No impl)...
        dw  DGROUP:_com_descon
        ; 19-Control E/S Gen,rico (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 20-No Definido (No impl)...
        dw  DGROUP:_com_descon
        ; 21-No Definido (No impl)...
        dw  DGROUP:_com_descon
        ; 22-No Definido (No impl)...
        dw  DGROUP:_com_descon
        ; 23-Trae Disposit. lógico (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 24-establ. Disposit.lógico (No impl)...
        dw  DGROUP:_pasa_sin_uso
        ; 25-Query de IOCTL (No impl)...
        dw  DGROUP:_pasa_sin_uso

_interrupcion   endp
```

```
;*********************************************
;      Fin del Frame...                       *
;*********************************************
_TEXT   ends
end
```

## Código fuente de : COMANDOS.C

```
/************************************
 * Código fuente de las rutinas de  *
 * atención a comandos...           *
 ************************************/
#include "frame.h"
#include "initdefs.h"
#include <dos.h>
#include <conio.h>
#include <bios.h>
#include <stdio.h>
#include <string.h>

/* Apuntador al Request Header... */
extern rh_t     far * rhptr ;
/* Dirección de la rutina Estrategia... */
extern void     estrategia (void) ;
/* Vector de BPB's... */
bpb_t           bpb1 [5] ;
/* Apuntador a BPB's requerido por DOS... */
bpb_t           * bpbs_arreglo [5] ;
/* Estructura con información acerca de los */
/* discos virtuales soportados... */
Disco_Virtual   Volumenes [5] ;
/* Total de discos a simular... */
WORD            Total_Discos ;
/* Guarda la linea de comandos... */
BYTE            Linea_Comando [120] ;

/* Guarda los parms. fisicos del disco duro... */
struct params_disco_fisico      d_i;
/* Información acerca del file system... */
struct file_system              f_s;
/* Número fisico del drive de trabajo... */
BYTE                            disp_boot;
/* Buffer para E/S de disco... */
BYTE            buf_tmp [ TAM_BUF_TMP ];
```

```
/* Rutinas para depuración de código fuente... */
#include "utileria.h"
/* Acceso de bajo nivel a disco... */
#include "accdisco.h"
/* Rutinas de atención a varios comandos... */
#include "varios.h"
/* Atención al comando Output... */
#include "output.h"
/* Atención al comando Input... */
#include "input.h"
/* Atención al comando de inicialización... */
#include "init.h"
```

## Código fuente de : FRAME.H

```
/************************************
 *   Código genérico usado como frame para   *
 * un Device Driver escrito en Ensamblador   *
 *   con interfaces en C...                  *
 ************************************/
#define BYTE        unsigned char
#define WORD        unsigned int
#define DWORD       unsigned long
#define CIERTO      1
#define FALSO       0
#define ERROR               0x8000
#define OCUPADO             0x0200
#define TERMINADO           0x0100
/*
Definiciones para códigos de error...
*/
#define PROT_ESCRITURA                  0
#define UNIDAD_DESCONOCIDA      1
#define DRIVE_NO_PREPARADO      2
#define DESCONOCIDO             3
#define ERROR_CRC                       4
#define LONG_RH_ERRONEA         5
#define ERROR_BUSQUEDA          6
#define MEDIA_DESCONOCIDA       7
#define SECTOR_NO_EXISTE        8
#define OUT_OF_PAPER            9
#define FALLA_ESCRITURA         10
#define FALLA_LECTURA           11
#define FALLA_GENERAL           12
#define CAMBIO_DISCO_INVALIDO   13
/************************************
 *Estructura para accesar el device header...   *
 ************************************/
typedef struct device_header_struct {
    struct device_header_struct far * sigdev ;
    WORD                            atributo ;
    void                ( * estrategia)  (void) ;
    void                ( * interrupcion) (void) ;
    BYTE                nombre [8] ;
} deviceheader_t ;
```

## Código fuente de : FRAME.H

```
/************************************
 *   Código genérico usado como frame para   *
 * un Device Driver escrito en Ensamblador   *
 *   con Interfaces en C...                  *
 ************************************/
#define BYTE        unsigned char
#define WORD        unsigned int
#define DWORD       unsigned long
#define CIERTO      1
#define FALSO       0
#define ERROR               0x8000
#define OCUPADO             0x0200
#define TERMINADO           0x0100
/*
Definiciones para códigos de error...
*/
#define PROT_ESCRITURA                  0
#define UNIDAD_DESCONOCIDA      1
#define DRIVE_NO_PREPARADO      2
#define DESCONOCIDO             3
#define ERROR_CRC                       4
#define LONG_RH_ERRONEA         5
#define ERROR_BUSQUEDA          6
#define MEDIA_DESCONOCIDA       7
#define SECTOR_NO_EXISTE        6
#define OUT_OF_PAPER            9
#define FALLA_ESCRITURA         10
#define FALLA_LECTURA           11
#define FALLA_GENERAL           12
#define CAMBIO_DISCO_INVALIDO   13
/************************************
 *Estructura para accesar el device header...   *
 ************************************/
typedef struct device_header_struct {
    struct device_header_struct far * sigdev ;
    WORD                                atributo ;
    void                    ( * estrategia)  (void) ;
    void                    ( * interrupcion) (void) ;
    BYTE                    nombre [8] ;
} deviceheader_t ;
```

```
/************************************
 * Estructura que define el BPB de un disco...  *
 ************************************/
typedef struct bpb_struct_struct {
      WORD          bps ;
      BYTE          spCl ;
      WORD          res_sectores ;
      BYTE          num_FATs ;
      WORD          archs_raiz ;
      WORD          tamano_vol ;
      BYTE          media_byte ;
      WORD          spFAT ;
      WORD          spt ;
      WORD          hpc ;
      DWORD ocultos ;
      DWORD tamano_vol_32 ;
} bpb_t ;

/*****************************
 * Parte fija del Request Header...  *
 *****************************/
typedef struct rhfixed_struct {
      BYTE    longitud ;
      BYTE    unidad  ;
      BYTE    comando ;
      WORD    status  ;
      BYTE    res [8] ;
} rh_t ;

/************************
 * Parte Variable : Comando 0. *
 ************************/
typedef struct rh_init_struct {
      rh_t          rh ;
      BYTE          nunidades ;
      void          ( far *  dir_break) (void);
      bpb_t         * far * bpbtab ;
      char          drive ;
} rh0_t ;
```

```
/**************************
 * Parte Variable : Comando 1.  *
 **************************/
typedef struct rh_media_check_struct {
      rh_t              rh ;
      BYTE              media ;
      BYTE              md_stat ;
      char far          * volid ;
} rh1_t ;

/**************************
 * Parte Variable : Comando 2.     *
 **************************/
typedef struct rh_get_bpb_struct {
      rh_t              rh ;
      BYTE              media ;
      char  far * buf  ;
      bpb_t far         * bpb ;
} rh2_t ;

/*********************************
 * Parte Variable : Comandos 3,12.        *
 *********************************/
typedef struct rh_ioctl_struct {
      rh_t              rh ;
      BYTE              media ;
      char far          * buf ;
      WORD              cantidad ;
      WORD              inicio ;
} rh3_t, rh12_t ;

/*****************************
 * Parte Variable : Comandos 4,8,9. *
 *****************************/
typedef struct rh_io_struct {
      rh_t              rh ;
      BYTE              media ;
      char far          * buf  ;
      WORD              cantidad ;
      WORD              inicio  ;
      char far          * volid;
} rh4_t, rh8_t, rh9_t ;
```

## Código fuente de : INITDEFS.H

```
/*********************************
 * PROGRAMA :                    *
 * Incluido para Inicialización de Driver  *
 *********************************/

/* Tamaño de Buffer Requerido para API */
#define TAM_BUF_TMP  512
/* Tamaño de Buffer Requerido para FATs */
#define TAM_BUF_FAT  512

/* Máximo de particiones en un disco duro */
#define MAXPART 4

/* # de Bytes Por Sector   */
#define TAM_SECTOR     512
#define TAM_ENTR_DIR   sizeof (struct renglon_dir)
/* # Entradas de dir/Sector */
#define DIRSPSEC (TAM_SECTOR/TAM_ENTR_DIR)

/* Status de nombre de archivo DOS */
#define DIREND         0x00
/* Cluster del Directorio Raiz    */
#define DIR_RAIZ       0
/* Primer Cluster Efectivo en área de Datos */
#define PRIMER_CLUSTER 2

/* System ID - Partición no utilizada   */
#define DOS_NO_USADO   0
/* System ID - FAT de 12-bits          */
#define DOS_FAT12      1
/* System ID - FAT de 16-bit           */
#define DOS_FAT16      4
/* System ID - Partición Extendida DOS  */
#define DOS_PART_EXT   5
/* System ID - Partición ver. 4.0 >32Mb */
#define DOS_40PAR      6
```

```
/***************************
 * Atributos de archivos   *
 ***************************/
#define R_O          0x01
#define OCULTO       0x02
#define SISTEMA      0x04
#define VOLUMEN      0x08
#define SUBDIRECTORIO 0x10
#define ARCHIVO      0x20
#define NO_USADO     0x00
#define BORRADO      0xE5
#define DIRECTORIO   0x2E

/***************************
 *  Parámetros de disco    *
 * dependientes de Hardware *
 ***************************/

struct params_disco_fisico {
  BYTE   drives;
  WORD   cabezas;
  WORD   cilindros;
  WORD   sectores;
};


/*******************************
 * Estructura que define al Boot DOS  *
 *******************************/
struct  BOOT_struct {
  /* Salto al código de arranque   */
  BYTE   entry_point [3];
  /* Nombre y Versión del fabricante */
  BYTE   oem [8];
  /* Características que definen al Disco */
  bpb_t  bpb;
};
```

```
/*************************
 * Definición de una entrada      *
 * de la Tabla de Partición       *
 *************************/
struct  renglon_part  {
  /* Indicador de Boot          */
  BYTE  boot_ID;
  /* Cabeza, Sector y Cil. del sector de arranque */
  BYTE  boot_HSC [3];
  /* Identif. del Sist. Op.     */
  BYTE  system_ID;
  /* Cabeza, Sector y Cil. del último sector     */
  BYTE  end_HSC [3];
  /* Distancia del Cero Físico  */
  DWORD sector_offset;
  /* Longitud en sectores       */
  DWORD sector_length;
};

/*****************************
 *      Estructura que define a la      *
 *      tabla de Partición del disco    *
 *****************************/
struct  particion {
  /* Código de arranque de disco */
  BYTE     codigo [446];
  /* Entradas de la tabla   */
  atruct   renglon_part p_tbl [MAXPART];
  /* Firma de Validez de Tabla   */
  WORD     firma;
};

/*************************
 * Unión que representa           *
 * una entrada de la FAT          *
 *************************/

union union_FAT {
 struct {
  unsigned int _16 : 16 ;
 } fat_16 ;
```

```
  struct {
    unsigned int _12 : 12;
    unsigned int xxx : 4;
  } fat_12_lo;

  struct {
    unsigned int xxx : 4;
    unsigned int _12 : 12;
  } fat_12_hi;

};

/**************************
 * Definición de una entrada      *
 * de Directorio                  *
 **************************/
struct  rengion_dir {
   BYTE    nombre [8];
   BYTE    ext [3];
   BYTE    atributo;
   BYTE    res [10];
   WORD    hora;
   WORD    fecha;
   WORD    dir_FAT;
   DWORD   tamano;
};
```

```
/*************************
 * Información Miscelanea      *
 * para cada File System       *
 *************************/

struct file_system {
  /* Drive del File System */
  BYTE      drive;
   /* Partición del File System*/
  BYTE      particion;
  /* # de Sector del inicio de Partición */
  DWORD     inicio;
   /* Sector base del Boot*/
  DWORD     base;
  /* # de Bits Por Celda de FAT */
  BYTE      bpFAT;
  /* Sectores Por Cluster */
  BYTE      spCl;
  /* Número de Sectores Reservados */
  WORD      res_sectores;
  /* # de FATs          */
   BYTE     num_FATs;
   /* # de archivos en Dir. Raiz */
  WORD      archs_raiz;
   /* # de Sectores Por FAT   */
  WORD      spFAT;
   /* # de Sectores ocultos   */
  DWORD     ocultos;
  /* # de celdas de FAT Por Sector   */
  WORD      eFATps;
   /* RBA del primer sector de FAT */
  DWORD     FAT_rba;
  /* RBA del primer sector de Directorio */
  DWORD     DIR_rba;
  /* RBA del primer sector de datos   */
  DWORD     DATA_rba;
  /* RBA de FAT actualmente en Buffer */
   DWORD    FAT_cargada;
   /* # de Sectores por FAT   */
  WORD      tamano_FAT;
  /* # de Sectorea por DIR   */
  WORD      tamano_DIR;
   /* # de Sectores por Volúmen */
   DWORD    tamano_VOL;
```

```
   /* Apuntador al buffer de FAT  */
   BYTE        *FAT_ptr;
} ;

/*************************************
 * Estructura para mantener información          *
 * acerca de la posición del                     *
 * Archivo-Disco Virtual en la FAT...            *
 *************************************/
struct Cadena_FAT {
      WORD Cluster ;
      WORD Cuantos ;
} ;

/*****************************************
 * Estructura para mantener informacién acerca                *
 * del Archivo-Disco Virtual en la FAT...                *
 *****************************************/
typedef struct D_V {
      BYTE Nombre [14] ;
        WORD Pos_Dir ;
        WORD Tot_Sects ;
      struct Cadena_FAT  Cadena_Virtual [40] ;
} Disco_Virtual ;
```

## Código fuente de : UTILERÍA.H

```
/*****************************************
 * Conjunto de utilerias para Depuración de código     *
 *****************************************/
void impr_entero( unsigned int entero) ;
void writechar(char ch, int ctos) ;
void writecharTELE(char ch) ;
void writecadena(char far * cad) ;
unsigned char lee_caracter(void) ;
void impr_entero( unsigned int entero) {
        unsignad char vals[20] ;
        int       actual = 0 ;
        do  {
                entero -= ( vals[actual] = entero % 10 ) ;
                vals[actual++] += '0' ;
                entero /= 10 ;
        }    while ( entero != 0 ) ;
        while ( --actual != -1 )  writecharTELE ( vals[actual] ) ;
}
```

```
void writechar(char ch, int ctos) {
  _AH = 0x09;
  _AL = ch;
  _BH = 0;
  _BL = 0xb0;
  _CX = ctos;
  geninterrupt(0x10);
}
void writecharTELE(char ch) {
  _AH = 0x0E;
  _AL = ch;
  _BH = 0;
  _BL = 0xb0;
  geninterrupt(0x10);
}
void writecadena(char far * cad) {
  for ( ; *cad != '\0'; cad ++ ) writecharTELE ( *cad ) ;
}
unsigned char lee_caracter(void) {
  _AH = 0x00;
  geninterrupt(0x16);
  return _AL ;
}
```

## Código fuente de : ACCDISCO.H

```
/****************************************
 *  RUTINAS DE ACCESO A DISCO REAL      *
 ****************************************/

/* Buffer para transferencia de sectores de FAT del disco real... */
BYTE   buf_FAT [ TAM_BUF_FAT ] ;

/**********************
 * Prototipos de Función... *
 **********************/

int biosdiscoEXT (int cmd, int drive, int cabeza,int pista,
                  int sector, int Tot_sects, void far *buffer);
WORD  Procesa_LBA  (WORD  cmd,WORD  drive,WORD  cuenta,void  far
*b_ptr,DWORD rba) ;
WORD Cluster_Valido (WORD cluster) ;
WORD Sig_Cluster (WORD cluster)  ;
WORD  Lee_Cluster(WORD  cluster,  BYTE  far  *  b_ptr,  BYTE  clstr_sec,  BYTE
cuenta);
WORD  Escribe_Cluster(WORD  cluster,BYTE  far  *b_ptr,BYTE  clstr_sec,BYTE
cuenta);
```

```
/*****************************************************
 * FUNCION: B i o s d i s c o E X T                  *
 * NOTA   : BiosdiscoEXT realiza la función indicada por *
 *  el parám. cmd. Es una extensión a la rutina de librería *
 *  biosdisk, que no acepta FarPointers. Fue modificada to- *
 *  mando como base el código binario de la rutina extraida *
 *  de la librería estandar de Turbo C...            *
 *****************************************************/
int biosdiscoEXT (int cmd, int drive, int cabeza,int pista,
     int sector, int Tot_Sects, void far *buffer)  {

     _AH =  cmd ;
     _AL =  Tot_Sects ;
     asm    mov BX, [ BP + 10h ] ;
     asm    mov ES, [ BP + 12h ] ;
     _CX =  pista ;
     _CX >>= 2 ;
     asm    AND CL,0C0h ;
     asm    ADD CL,[BP+0Ch]
     _CH =  pista ;
     _DH =  cabeza ;
     _DL =  drive ;
     asm    int 13h ;
     return _AH ;
}
/*****************************************************
 * FUNCION: P r o c e s a L B A                      *
 * NOTA   : Procesa_LBA realiza una lectura o escritura *
 *       de/hacia el sector lógico rba indicado de un *
 *       file system REAL de DOS...                  *
 *****************************************************/
WORD Procesa_LBA ( WORD cmd, WORD drive, WORD cuenta,
           void far * b_ptr, DWORD rba)
{
  WORD rdo;
  WORD cil;
  WORD cabeza;
  WORD sector;

  cil    = ((rba / d_i.sectores) / d_i.cabezas);
  cabeza = ((rba / d_i.sectores) % d_i.cabezas);
  sector = ((rba % d_i.sectores) + 1);
  rdo = biosdiscoEXT (cmd, drive, cabeza, cil, sector, cuenta, b_ptr ) ;
  return rdo;
}
```

```
/*******************************************************
 * FUNCION: C l u s t e r _ V a l i d o                *
 * Checa si el Cluster indicado es válido en el file system real *
 *******************************************************/
WORD Cluster_Valido (WORD cluster) {
  if ((cluster < PRIMER_CLUSTER)              ||
     ((f_s.bpFAT == 16) && (cluster > 0xFFF8 )) ||
     ((f_s.bpFAT == 12) && (cluster > 0x0FF8)))
   return FALSO ;

  return CIERTO ;
}

/*******************************************************
 * FUNCION   S i g _ C l u s t e r                     *
 * Indica el cluster que sigue al parámetro dentro de una cadena *
 * en la table de FAT                                  *
 *******************************************************/
WORD   Sig_Cluster (WORD cluster) {

 WORD   rdo;
 WORD   n_cluster;
 WORD   fat_pos;
 DWORD  fat_rba;
 DWORD  l_cluster;
 union union_FAT *f_ptr;

 l_cluster = cluster;
 if (f_s.bpFAT == 12) {
   l_cluster = ((l_cluster * 3) / 2);
   fat_pos = (l_cluster % TAM_SECTOR);
   fat_rba = (f_s.FAT_rba + (l_cluster / TAM_SECTOR));
   if (f_s.FAT_cargada != fat_rba) {
     f_s.FAT_cargada = fat_rba;
     Procesa_LBA (0x02, disp_boot, 1, f_s.FAT_ptr, fat_rba);
   }

   f_ptr = (union union_FAT *) (f_s.FAT_ptr + fat_pos);
   if (cluster & 1)
     n_cluster = f_ptr->fat_12_hi._12;
   else
     n_cluster = f_ptr->fat_12_lo._12;
 }
```

```
 else {
     l_cluster      = (l_cluster * 2);
     fat_pos = (l_cluster % TAM_SECTOR);
     fat_rba = (f_s.FAT_rba + (l_cluster / TAM_SECTOR));
     if (f_s.FAT_cargada != fat_rba) {
       f_s.FAT_cargada = fat_rba;
       Procesa_LBA (0x02, disp_boot, 1, f_s.FAT_ptr, fat_rba);
     }

     f_ptr = (union union_FAT *) (f_s.FAT_ptr + fat_pos);
     n_cluster = f_ptr->fat_16._16;
 }

 if (! Cluster_Valido (n_cluster))  n_cluster = 0xFFFF;
 return n_cluster;
}

/*****************************************************
 * FUNCION:  L e e _ C l u s t e r                   *
 * Lee "cuenta" sectores a partir del cluster "cluster" y sector  *
 * secuencial "clstr_sec". Guarda el resultado en  "b_ptr".       *
 *****************************************************/
WORD Lee_Cluster (WORD cluster, BYTE far * b_ptr,
                  BYTE clstr_sec, BYTE cuenta)
{
 WORD  rdo = 0x00 ;
 DWORD rba;
 DWORD l_cluster;

 l_cluster = cluster;
 if (Cluster_Valido (cluster)) {
   rba = (((l_cluster - PRIMER_CLUSTER) * f_s.spCl) + f_s.DATA_rba);
   rba += clstr_sec;
   rdo = Procesa_LBA (0x02, disp_boot, cuenta, b_ptr, rba);
 }
 else  rdo = 0xFFFF;
 return rdo;
}
```

```
/*****************************************************
 * FUNCION:  E s c r i b a _ C l u s t e r                    *
 * Escribe "cuenta" sectores a partir del cluster "cluster" y sector  *
 * secuencial "clstr_sec". Lee la cadena a escibir de "b_ptr".        *
 *****************************************************/
WORD Escribe_Cluster (WORD cluster, BYTE far * b_ptr,
                      BYTE clstr_sec, BYTE cuenta)
{
 WORD  rdo = 0x00 ;
 DWORD rba;
 DWORD l_cluster;

 l_cluster = cluster;
 if (Cluster_Valido (cluster))  {
   rba = (((l_cluster - PRIMER_CLUSTER) * f_s.spCl) + f_s.DATA_rba);
   rba += clstr_sec;
   rdo = Procesa_LBA (0x03, disp_boot, cuenta, b_ptr, rba);
 }
 else  rdo = 0xFFFF;
 return rdo;
}
```

## Código fuente de :  VARIOS.H

```
/*************************************************
 * VARIOS.H - Rutinas de respuesta a algunos comandos     *
 *************************************************/

/***************************
 *       Prototipos de función :      *
 ***************************/
void com_descon   (void) ;
void mediacheck   (void) ;
void getbpb       (void) ;
void pasa_sin_uso (void) ;

/*************************************************
 *       Atención a comandos que no se debieran recibir...     *
 *************************************************/
void com_descon (void) {
   /* Avisa de comando desconocido...*/
   rhptr -> status = TERMINADO | ERROR | DESCONOCIDO ;
}
```

```
/*************************************
 *  Atención al comando 1 : Media_Check...     *
 *************************************/
void mediacheck (void) {
   rh1_t far * rh1 ;
   /* Apunta rh1 al Request Header... */
   rh1 = ( rh1_t far * ) rhptr ;
   /* Indica que el volúmen no ha cambiado... */
   rh1 -> media = 1 ;
   rh1 -> rh.status = TERMINADO ;
}

/*************************************
 *       Atención al comando 2 : GetBPB...       *
 *************************************/
void getbpb (void) {
   rh2_t far * rh2 ;
   WORD   rdo ;
   struct BOOT_struct * boot ;
   WORD   i ;
   /* Apunta rh1 al Request Header... */
   rh2 = ( rh2_t far * ) rhptr ;
      rdo  =  Lee_Cluster  (Volumenes[rh2->rh.unidad].Cadena_Virtual[0].Cluster,
buf_tmp, 0, 1 );
   if ( rdo != 0x00 ) {
      writecadena("Error en Lectura de Boot ...\n\r");
      rh2 -> rh.statue = TERMINADO | ERROR | FALLA_LECTURA ;
      goto Fin ;
   }
   boot = ( struct BOOT_struct * ) buf_tmp ;
   /* Inicializa el (los) BPB ('s)*/
   bpb1[rh2->rh.unidad].bps =        boot->bpb.bps ;
   bpb1[rh2->rh.unidad].spCl =       boot->bpb.spCl ;
   bpb1[rh2->rh.unidad].res_sectores = boot->bpb.res_sectores ;
   bpb1[rh2->rh.unidad].num_FATs =    boot->bpb.num_FATs ;
   bpb1[rh2->rh.unidad].archs_raiz =  boot->bpb.archs_raiz ;
   bpb1[rh2->rh.unidad].tamano_vol =  boot->bpb.tamano_vol ;
   bpb1[rh2->rh.unidad].media_byte =  boot->bpb.media_byte ;
   bpb1[rh2->rh.unidad].spFAT =      boot->bpb.spFAT ;
   bpb1[rh2->rh.unidad].spt =        boot->bpb.spt ;
   bpb1[rh2->rh.unidad].hpc =        boot->bpb.hpc ;
   bpb1[rh2->rh.unidad].ocultos =    boot->bpb.ocultos ;
   bpb1[rh2->rh.unidad].tamano_vol_32 =boot->bpb.tamano_vol_32 ;
   rh2 -> buf = ( char far * ) &bpb1[rh2->rh.unidad] ;
```

```
   /* Indica la dirección del vector de BPB's... */
   /*rh2 -> bpb = ( bpb_t far * ) bpbs_arreglo ;*/
   rh2 -> bpb = ( bpb_t far * ) &bpb1[rh2->rh.unidad] ;

   rh2 -> rh.status = TERMINADO ;
   Fin : ;
}
/*************************************
 *      Atención a comandos no importantes    *
 *      para la aplicación...                 *
 *************************************/
void pasa_sin_uso (void) {
   rhptr -> status = TERMINADO ;
}
```

## Código fuente de : OUTPUT.H

```
/*****************************************
 * Rutina de atención al comando O u t p u t          *
 *****************************************/
void output (void) ;

void output (void) {
   WORD Inicio, Ctos ;
   BYTE far * Buf_Out ;
   WORD Clst_Rel, Cluster, Secuencia, Continuos ;
   WORD a_escribir ;
   WORD rdo ;
   BYTE I ;
   rh8_t far * rh8 ;

   /* Apunta rh1 al Request Header... */
   rh8 = ( rh8_t far * ) rhptr ;

   /* Si el sector es CONOCIDO : */
   if ( rh8 -> Inicio <= Volumenes[rh8->rh.unidad].Tot_Sects - 1 ) {
      Inicio =  rh8 -> Inicio ;
      Ctos   =  rh8 -> cantidad ;
      Buf_Out = rh8 -> buf ;

      for(i=0;
            Inicio>Volumenes[rh8->rh.unidad].Cadena_Virtual[i].Cuantos*f_s.spCl-
1;i++)
        Inicio -= Volumenes[rh8->rh.unidad].Cadena_Virtual[i].Cuantos * f_s.spCl ;
```

```
      Clst_Rel = Inicio / f_s.spCl ;
      Cluster  = Volumenes[rh8->rh.unidad].Cadena_Virtual[i].Cluster + Clst_Rel ;
      Secuencia= Inicio % f_s.spCl ;
      Continuos=(Volumenes[rh8->rh.unidad].Cadena_Virtual[i].Cuantos-Clst_Rel);
      Continuos*= f_s.spCl - Secuencia ;

      for ( a_escribir = 0; Ctos > 0; Buf_Out += a_escribir * f_s.spCl ) {
         a_escribir = ( Ctos > Continuos ) ? Continuos : Ctos ;
         rdo = Escribe_Cluster ( Cluster, Buf_Out, Secuencia, a_escribir);
         if ( rdo != 0x0000 ) {
             rhptr -> status = TERMINADO | ERROR | FALLA_ESCRITURA ;
             goto Fin ;
         }
         Ctos -= a_escribir ;
         Cluster = Volumenes[rh8->rh.unidad].Cadena_Virtual[++i].Cluster ;
         Secuencia = 0 ;
                                                    Continuos=Volumenes[rh8-
>rh.unidad].Cadena_Virtual[++i].Cuantos*f_s.spCl
      }

      rh8 -> rh.status = TERMINADO ;
      goto Fin ;
   }
   else {
     rh8 -> rh.status = TERMINADO | ERROR | FALLA_ESCRITURA ;
     goto Fin ;
   }
   Fin : ;
}
```

## Código fuente de : INPUT.H

```
/*****************************************
 * Rutina de atención al comando : I n p u t          *
 *****************************************/
void input (void)  {

  WORD Inicio, Ctos ;
  BYTE far * Buf_Inp ;
  WORD Clst_Rel, Cluster, Secuencia, Continuos ;
  WORD a_leer ;
  WORD rdo ;
  BYTE i ;
  rh4_t far * rh4 ;
```

```
 /* Apunta rh4 al Request Header... */
 rh4 = ( rh4_t far * ) rhptr ;

 Inicio =  rh4 -> inicio ;
 Ctos   =  rh4 -> cantidad ;
 Buf_Inp = rh4 -> buf ;

 if ( Inicio > Volumenes[rh4->rh.unidad].Tot_Sects -1 ) {
   rhptr -> status = TERMINADO | ERROR | SECTOR_NO_EXISTE ;
   goto Fin ;
 }

 for ( i=0; Inicio > Volumenes[rh4->rh.unidad].Cadena_Virtual[i].Cuantos*f_s.spCl-
1; i++)
     Inicio -= Volumenes[rh4->rh.unidad].Cadena_Virtual[i].Cuantos * f_s.spCl ;

 Clst_Rel = Inicio / f_s.spCl ;
 Cluster  = Volumenes[rh4->rh.unidad].Cadena_Virtual[i].Cluster + Clst_Rel ;
 Secuencia= Inicio % f_s.spCl ;
  Continuos= ( Volumenes[rh4->rh.unidad].Cadena_Virtual[i].Cuantos - Clst_Rel) *
f_s.spCl
                - Secuencia ;

 for ( a_leer = 0; Ctos > 0; Buf_Inp += a_leer * f_s.spCl ) {
   a_leer = ( Ctos > Continuos ) ? Continuos : Ctos ;
   rdo = Lee_Cluster ( Cluster, Buf_Inp, Secuencia, a_leer ) ;
   if ( rdo != 0x0000 ) {
     rhptr -> status = TERMINADO | ERROR | FALLA_LECTURA ;
     goto Fin ;
   }

   Ctos -= a_leer ;
   Cluster = Volumenes[rh4->rh.unidad].Cadena_Virtual[++i].Cluster ;
   Secuencia = 0 ;
   Continuos = Volumenes[rh4->rh.unidad].Cadena_Virtual[++i].Cuantos * f_s.spCl
;
 }

 rhptr -> status = TERMINADO ;

Fin : ;

}
```

## Código fuente de : INIT.H

```
/*********************************************************
 * CÓDIGO   : Rutinas de respuesta al comando : Inicialización...     *
 *********************************************************/

void inicializa (void) ;
WORD Lee_File_System (WORD drive, DWORD rba, WORD p_base) ;
WORD Inicializa_FS (WORD c, WORD h, WORD s) ;
WORD Busca_Archivo (BYTE *n_ptr,struct renglon_dir *d_ptr) ;
WORD Busca_Entrada (WORD cluster, struct renglon_dir *d_ptr) ;
void lee_linea_coms  ( void ) ;
WORD scan_linea_coms ( void ) ;
void muestra_params  ( void ) ;


/*********************************
 * FUNCION: I n i c i a l i z a                *
 *********************************/
void inicializa (void) {
  WORD rdo;
  BYTE cEslabon ;
  WORD  nCtosVan, nClusterInicio, Sig_Cl1, Sig_Cl2 ;
  struct renglon_dir         renglonDir;
  struct BOOT_struct          * boot ;
  rh0_t far * rh0 ;
  int i, Drive_En_Turno ;

/*****************************
 *Apunta rh0 al Request Header... *
 *****************************/
  rh0 = ( rh0_t far * ) rhptr ;
/*********************************
 * Carga información del Disco Real...     *
 *********************************/
  f_s.FAT_ptr = buf_FAT ;
  disp_boot = 0x80;
  if ( Lee_File_System (disp_boot, 0x0L, 0x0) )      {
    writecadena("\n\rError en Lectura en tabla de partición de disco C:. Simulador
Abortado...\n\r");
    return ;
  }

  f_s.FAT_cargada = f_s.FAT_rba;
```

```
 rdo = Process_LBA (0x02, disp_boot, 1, f_s.FAT_ptr, f_s.FAT_rba);
 if ( rdo != 0x00 ) {
   writecadena("\n\rError en Lectura de FAT. Simulador Abortado...\n\r") ;
   return ;
 }


/*********************************************
 * Lee la linea de comando y configura la inicialización...      *
 *********************************************/
 writecadena ( "\n\n      *** Simulador Multidisco v1.0 ***\n\r" ) ;
 lee_linea_coms () ;
 if ( scan_linea_coms () )  {
   writecadena ( "\n\rError en linea de comandos. Simulador Abortado...\n\r" ) ;
   return ;
 }
 else {
   muestra_params () ;
   writecadena ( "\n\r" ) ;
 }

  for ( Drive_En_Turno = 0; Drive_En_Turno < Total_Discos; Drive_En_Turno ++ )
{
   /* Lee la información de directorio del Archivo-Disco Virtual... */
   rdo = Busca_Archivo (Volumenes[Drive_En_Turno].Nombre, &renglonDir) ;
   if ( rdo != 0x00 ) {
     writecadena ( "\n\rError : Archivo " ) ;
     writecadena ( Volumenes[Drive_En_Turno].Nombre ) ;
     writecadena ( " no encontrado. Simulador Abortado...\n\r" );
     return ;
   }
   /* Genera la Cadena descriptore de FAT del Archivo-Disco Virtual... */
   Volumenes[Drive_En_Turno].Cadena_Virtual [0].Cuantos = 1 ;
          Volumenes[Drive_En_Turno].Cadena_Virtual  [0].Cluster  =  Sig_Cl1  =
renglonDir.dir_FAT ;
   Sig_Cl2 = Sig_Cluster(Sig_Cl1) ;
   for ( cEslabon = 0; Sig_Cl2 != 0xFFFF; Sig_Cl2 = Sig_Cluster(Sig_Cl1) ) {
     if ( Sig_Cl2-1 == Sig_Cl1 ) {
       Volumenes[Drive_En_Turno].Cadena_Virtual [cEslabon].Cuantos ++ ;
       Sig_Cl1 ++ ;
     }
```

```
    else {
      cEslabon ++ ;
      Volumenes[Drive_En_Turno].Cadena_Virtual [cEslabon].Cluster = Sig_Cl2 ;
      Volumenes[Drive_En_Turno].Cadena_Virtual [cEslabon].Cuantos = 1 ;
      Sig_Cl1 = Sig_Cl2 ;
    }
  }
  /* Indica el tamaño inicial del Archivo-Disco Virtual... */
  Volumenes[Drive_En_Turno].Tot_Sects = rengionDir.tamano / TAM_SECTOR ;
  /* Lee el primer sector del Archivo-Disco Virtual para retornar el BPB. */
     rdo  =  Lee_Cluster  (Volumenes[Drive_En_Turno].Cadena_Virtual[0].Cluster,
buf_tmp, 0, 1 );
  if ( rdo != 0x00 ) {
    writecadena ( "\n\rError en Lectura de Boot de archivo " ) ;
    writecadena ( Volumenes[Drive_En_Turno].Nombre ) ;
    writecadena ( ". Simulador Abortado...\n\r" );
    return ;
  }
  boot = ( struct BOOT_struct * ) buf_tmp ;
  writecadena ( "Tamaño de " ) ;
  writecadena ( Volumenes[Drive_En_Turno].Nombre ) ;
  writecadena ( " en KiloBytes-->\a\a" ) ;
  impr_entero ( Volumenes[Drive_En_Turno].Tot_Sects / 2 ) ;
  writecadena ( "\n\r" ) ;
  /* Inicializa los BPB ('s)...  */
  bpb1 [Drive_En_Turno]. bps =         boot->bpb.bps ;
  bpb1 [Drive_En_Turno]. spCl =        boot->bpb.spCl ;
  bpb1 [Drive_En_Turno]. res_sectores = boot->bpb.res_sectores ;
  bpb1 [Drive_En_Turno]. num_FATs =    boot->bpb.num_FATs ;
  bpb1 [Drive_En_Turno]. archs_raiz =  boot->bpb.archs_raiz ;
  bpb1 [Drive_En_Turno]. tamano_vol =  boot->bpb.tamano_vol ;
  bpb1 [Drive_En_Turno]. media_byte =  boot->bpb.media_byte ;
  bpb1 [Drive_En_Turno]. spFAT =       boot->bpb.spFAT ;
  bpb1 [Drive_En_Turno]. spt =         boot->bpb.spt ;
  bpb1 [Drive_En_Turno]. hpc =         boot->bpb.hpc ;
  bpb1 [Drive_En_Turno]. ocultos =     boot->bpb.ocultos ;
  bpb1 [Drive_En_Turno]. tamano_vol_32 =boot->bpb.tamano_vol_32 ;
}

/* Inicializa el vector de apuntadores a BPB's... */
for ( i=0; i < 5; i++ ) bpbs_arreglo[i] = & bpb1 [i] ;

/* Indica la dirección del vector de BPB's... */
rh0 -> bpbtab = bpbs_arreglo ;
```

```
 /* Retorna el número de Dispositivos a manejar... */
 rh0 -> nunidades = Total_Discos ;

 /* Indica la Dirección del fin de código... */
 rh0 -> dir_break = ( void ( far * ) (void)) inicializa ;
 (unsigned int) rh0 -> dir_break += (unsigned int) estrategia ;

 /* Avisa...*/
 writecadena ("\a\aInicialización de Simulador terminada...\n\r\a\a") ;

 rh0 -> rh.status = TERMINADO ;
}

/*********************************************************
 * Rutinas para Inicialización de Estructuras Descriptoras de      *
 * del Disco Real...                                               *
 *********************************************************/

/************************************************
 * FUNCION : Lee_File_System                    *
 * ENTRADAS : drive "drive" en que el File System está.  *
 * Sector "rba" en que inicia el File System             *
 * "p_base", Localización del sector base de la Partición.  *
 ************************************************/
WORD Lee_File_System (WORD drive, DWORD rba, WORD p_base)  {
  WORD                  rdo;
  WORD                  p_num;
  WORD                  cil;
  WORD                  cabeza;
  WORD                  sect;
  BYTE                  *c_ptr;
  DWORD                 ep_off;
  struct particion      *p_ptr;
  struct  region_part   *pe_ptr;

  rdo = biosdiscoEXT (0x02, drive, 0x00, 0x00, 0x01, 0x01, buf_tmp);
  if (rdo != 0x00)  return 0xFFFF;

  p_ptr = ( struct particion * ) buf_tmp;
  if (p_ptr->firma != 0xAA55)  return 0xFFFF;

  for (p_num = 0; p_num < MAXPART; p_num++)  {
    pe_ptr = &(p_ptr->p_tbl [p_num]);
    if (pe_ptr->boot_ID == 0x80)  break;
  }
```

```
  if ( p_num == MAXPART ) return 0xFFFF;

  switch (pe_ptr->system_ID) {
   case DOS_NO_USADO :
   case DOS_PART_EXT :
     return 0xFFFF;
   case DOS_FAT12 :
   case DOS_FAT16 :
   case DOS_40PAR : {
     f_s.drive= drive;
     f_s.particion    = p_base + p_num;
     f_s.inicio       = rba + pe_ptr->sector_offset;
     f_s.base = rba;
     if (pe_ptr->system_ID == DOS_FAT12) {
       f_s.bpFAT    = 12;
       f_s.eFATps   = 342;
     }
     else {
       f_s.bpFAT    =16;
       f_s.eFATps   =256;
     }
     c_ptr = &(pe_ptr->boot_HSC [0]);
     cabeza  = *c_ptr;
     sect    = (*(c_ptr + 1) & 0x3F);
     cil     = (*(c_ptr + 1) & 0xC0);
     cil     = (*(c_ptr + 2) | (cil << 2));
     if ( inicializa_FS (cil, cabeza, sect) )  return 0xFFFF;
     break;
   }
   default :
     return 0xFFFF;
  }
  return 0x00;
}
```

```
/*************************************************
 * FUNCTION: I n i c i a l i z a _ F S                    *
 *                                                        *
 * NOTA   : Inicializa_FS Lee el boot sector y establece  *
 *        algunos valores en la estructura file system (f_s).  *
 *************************************************/
WORD Inicializa_FS (WORD c, WORD h, WORD s) {
 WORD      rdo;
 DWORD     tamano_vol;
 struct BOOT_struct *b_ptr;

 rdo = biosdiscoEXT (0x02, f_s.drive, h, c, s, 1, buf_tmp ) ;
 if ( rdo != 0x00 ) {
   rdo = biosdiscoEXT (0x02, f_s.drive, h, c, s, 1, buf_tmp ) ;
   if ( rdo != 0x00 ) return 0xFFFF;
 }

 b_ptr = ( struct BOOT_struct * ) buf_tmp ;

 d_l.drives    = 1;
 d_l.cabezas = b_ptr->bpb.hpc;
 tamano_vol = (( b_ptr->bpb.tamano_vol == 0 ) ?
                    b_ptr->bpb.tamano_vol_32 : b_ptr->bpb.tamano_vol);
 d_l.cilindros       = (((tamano_vol / b_ptr->bpb.spt) / b_ptr->bpb.hpc) + 1) ;
 d_l.sectores        = b_ptr->bpb.spt;

 f_s.spCl            = b_ptr->bpb.spCl;
 f_s.res_sectores    = b_ptr->bpb.res_sectores;
 f_s.num_FATs        = b_ptr->bpb.num_FATs;
 f_s.archs_raiz      = b_ptr->bpb.archs_raiz;
 f_s.spFAT           = b_ptr->bpb.spFAT;
 f_s.ocultos         = b_ptr->bpb.ocultos;

 f_s.FAT_rba         = (f_s.base + f_s.ocultos + f_s.res_sectores);
 f_s.DIR_rba         = (f_s.FAT_rba + (f_s.num_FATs * f_s.spFAT)) ;
 f_s.DATA_rba        = (f_s.DIR_rba + ((f_s.archs_raiz+DIRSPSEC-1) /DIRSPSEC)) ;
 f_s.tamano_VOL      = tamano_vol;
 f_s.tamano_FAT      = f_s.DIR_rba -  f_s.FAT_rba;
 f_s.tamano_DIR      = f_s.DATA_rba - f_s.DIR_rba;

 return 0x00;
}
```

```
/***********************************************
 * Rutinas para Búsqueda del Archivo-Disco Virtual en el File*
 * System Anfitrión...                                       *
 ***********************************************/


/*********************************
 * FUNCION:  B u s c a _ A r c h i v o     *
 *********************************/
WORD Busca_Archivo (BYTE *n_ptr,struct region_dir *d_ptr) {
 WORD       i;
 WORD       rdo;
 WORD       cl;
 BYTE               *ptr;
 BYTE               *c_ptr;

 /*  Pasa a mayúsculas */
 for (c_ptr = n_ptr; *c_ptr; *c_ptr++ = (((*c_ptr >= 'a') && (*c_ptr <= 'z')) ?
                        (*c_ptr - 'a' + 'A') : (*c_ptr))) ;

 for (ptr = n_ptr, cl = DIR_RAIZ, i = 0; *ptr; i++)  {
  /* Elimina \'s */
  for ( ; *ptr == '\\'; ptr++)  ;
  for (c_ptr = d_ptr->nombre;c_ptr < (d_ptr->nombre + 8);*c_ptr++ = ' ') ;
  for (c_ptr = d_ptr->ext; c_ptr < (d_ptr->ext + 3); *c_ptr++ = ' ') ;
  for (c_ptr = d_ptr->nombre;((*ptr != '.') && (*ptr != '\\') && *ptr);
      *c_ptr++ = *ptr++) ;

  if (*ptr == '.') {
    ptr++;
    for (c_ptr = d_ptr->ext; ((*ptr != '\\') && *ptr); *c_ptr++ = *ptr++) ;
  }

  rdo = Busca_Entrada (cl, d_ptr);
  if (rdo == 0xFFFF)  break ;
  cl = d_ptr->dir_FAT;
 }
 rdo = ((i == 0) ? 0xFFFF : rdo);
 return rdo;
}
```

```
/*************************************************
 * FUNCION: B u s c a E n t r a d a                *
 *                                                 *
 * NOTAS :  Busca_Entrada encuentra la entrada de direc-  *
 * torio para el nombre de archivo especificado...        *
 *************************************************/
WORD Busca_Entrada (WORD cluster, atruct rengion_dir *d_ptr) {
 WORD        rdo;
 WORD        i;
 WORD        j;
 WORD        k;
 WORD        encontrado;
 BYTE       *fte;
 BYTE       *dst;
 BYTE       *fc_ptr;
 struct rengion_dir   *fd_ptr;

 for (i = 0; i < f_s.tamano_DIR; i++) {
    rdo = Procesa_LBA (0x02, disp_boot, 1, buf_tmp, (f_s.DIR_rba + i));
    if (rdo)  return 0xFFFF;
                for  (fc_ptr=  buf_tmp;  fc_ptr<buf_tmp+TAM_SECTOR;fc_ptr  +=
TAM_ENTR_DIR) {
       if (*fc_ptr == DIREND)  return 0xFFFF;
       fd_ptr = (struct rengion_dir *) fc_ptr;
       if (!(fd_ptr->atributo & VOLUMEN)) {
         fte = fd_ptr->nombre;
         dst = d_ptr ->nombre;
         for (j = 0, encontrado = CIERTO; j < 11; j++) {
            if (*(fte + j) != *(dst + j)) {
              encontrado = FALSO;
              break;
            }
         }
         if (encontrado) {
           for (j = 0;j < sizeof(struct rengion_dir); j++)
             *(dst + j) = *(fte + j) ;
           return cluster;
         }
       }
    }
 }
 return 0xFFFF;
}
```

```
/*************************************************
 * FUNCION: l a e _ l l n e a _ c o m s                          *
 *                                                               *
 * NOTAS :  Obtiene la linea de comandos a partir del archi-*
 *          vo config.sys y la guarda en Linea_Comando...        *
 *************************************************/
void lee_linea_coms  ( void ) {
  rh0_t far * rh0 ;
  int i, j ;

  /* Apunta rh0 al Request Header... */
  rh0 = ( rh0_t far * ) rhptr ;

  /* Obtiene la linea de comandos a partir del config.sys... */
  for (j = 0; * ((char far *)(rh0 -> bpbtab) + j) != ' '; j++ ) ;
  for ( i = 0, j ++; * ( (char far *) (rh0 -> bpbtab) + j ) != 10; j++, i++ )
    Linea_Comando [ i ] = * ((char far *)(rh0 -> bpbtab) + j);
  Linea_Comando [ i ] = '\0' ;
}


/*************************************************
 * FUNCION: s c a n _ l l n e a _ c o m s                        *
 *                                                               *
 * NOTAS :  Realiza scan sobre la linea de comandos y con-*
 *          figura las variables Volumenes y Total_Discos.       *
 *************************************************/
WORD scan_linea_coms ( void ) {

  int TotArchs ;
  BYTE * b = Linea_Comando ;
  int car_act, car_ext, contador ;

  /* Inicia el Scaneo de la linea de comandos... */
  Total_Discos = 0 ;
  TotArchs = 0;

  while (  *b != '\0' ) {
    for ( ; *b != '/' && *b != '\0'; b ++ ) ;
    if ( *b == '\0' )  break ;

    b ++ ;
    if ( *b == 'd' || *b == 'D' ) {   /* Parámetro Total de discos ...*/
        if ( * ( ++ b ) != ':' ) {
```

```
        writecadena("Error en linea de comandos en parámetro /d...\n\r");
          return 1 ;
        }
        for ( Total_Discos = 0, b ++; *b >= '0' && *b <= '9'; b ++ )
          Total_Discos = Total_Discos * 10 + *b - '0' ;
    }
    else  if ( *b == 'a' || *b == 'A' )  { /* Parámetro nombre de archivo ...*/
        if ( * ( ++ b ) != ':' ) {
        writecadena("Error en linea de comandos en parámetro /a...\n\r");
          return 1 ;
        }

        for ( b++, car_act = 0;
            ( ( *b >= '0' && *b <= '9' ) ||
                ( *b >= 'a' && *b <= 'z' ) ||
                ( *b >= 'A' && *b <= 'Z' ) ||
           *b == '-' || *b == '_' || *b == '$' || *b == '\''
            )  &&   car_act < 8 ;
            b ++
          )
          Volumenes [ TotArchs ] . Nombre [ car_act ++ ] = *b ;

        if ( *b == '.' ) {
          Volumenes [ TotArchs ] . Nombre [ car_act ++ ] = '.' ;
          for ( b++, car_ext = 0;
                ( ( *b >= '0' && *b <= '9' ) ||
                  ( *b >= 'a' && *b <= 'z' ) ||
                  ( *b >= 'A' && *b <= 'Z' ) ||
             *b == '-' || *b == '_' || *b == '$' || *b == '\''
                ) &&   car_ext < 3 ;
                b ++, car_ext ++
            )
            Volumenes [ TotArchs ] . Nombre [ car_act ++ ] = *b ;
        }
        Volumenes [ TotArchs ] . Nombre [ car_act ] = '\0' ;

        TotArchs ++ ;
    }
    else  {  /* Parámetros no reconocidos ...*/
      writecadena("Error en linea de comandos. parámetro desconocido : ");
      writecharTELE ( * b ) ;
      writecadena ( "\n\r" ) ;
      return 1 ;
    }
}
```

```
  if ( Total_Discos == 0 || Total_Discos > 5 ) Total_Discos = 5 ;
  if ( TotArchs > Total_Discos ) TotArchs = Total_Discos ;
  if ( TotArchs < Total_Discos ) Total_Discos = TotArchs;
  for ( contador = TotArchs; contador < Total_Discos; contador ++ )
    Volumenes [ contador ] . Nombre [ 0 ] = '\0' ;

  return 0 ;
}


/************************************************
 * FUNCION: m u e s t r a _ p a r a m s         *
 *                                              *
 * NOTAS :  Realiza salida formateada de la configuración *
 *          inicial del controlador...          *
 ************************************************/

void muestra_params ( void ) {

  BYTE cont, cont2 ;
  BYTE buffer [50] ;
  BYTE buffer2 [30] ;

  writecadena ( "\n\r    Los parámetros de instalación son :\n\r" ) ;
  writecadena ( Linea_Comando ) ;
  writecadena ( "\n\r" ) ;

  writecadena ("         +----------------------------+\n\r") ;
  writecadena ("         | Disco | Nombre del archivo |\n\r") ;
  writecadena ("         +-------+--------------------+\n\r") ;
  for ( cont=1; cont <= Total_Discos; cont++ ) {
     strcpy ( buffer, "         |       |                    |\n\r" ) ;
     * (buffer+16) = cont + '0' ;
          strcpy  (  buffer2,  (  Volumenes[cont-1].Nombre[0]  )  ?  Volumenes[cont-
1].Nombre : "No asignado..." ) ;
     for ( cont2 = 0; * ( buffer2+cont2 ) ; cont2++ )
     {     * ( buffer+ 21 + cont2) = ( * ( buffer2 + cont2 ) ) ;  }
     writecadena ( buffer) ;
     if ( cont == Total_Discos )
          writecadena (  "         +----------------------------+\n\r" ) ;
     else writecadena ( "         +-------+--------------------+\n\r" ) ;
  }
}
```

# Apéndice B. Código fuente de la utilería ARRANGE.

La función de la utilería ARRANGE ya ha sido mencionada antes en este trabajo, en el capítulo dedicado a describir la implementación del proyecto. A continuación se muestra su código fuente :

```
/*****************************************************************************
 *  Programa : ARRANGE.C                                                     *
 *  Reemplaza cadenas dentro de un archivo ASCII basado en                   *
 *  Las reglas contenidas en un archivo guía...                              *
 *      Formato :                                                            *
 *  ARRANGE <Arch.Guía> <ArchivoEntrada> <ArchivoSalida>                     *
 *****************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*****************************************
 *       Prototipos de función...        *
 *****************************************/
void read_cmds ( char *cmds[], int limit, FILE *fp_cmds ) ;
void run_cmds ( char *cmds[], FILE *fp_in, FILE *fp_out ) ;
void substitute ( char *a, char *line ) ;
void error ( char *s1, char *s2 ) ;
void apply_cmds ( char *cmds[], char *line ) ;

#define MAXCMDS   100
#define MAXLINE   100

char *progname ;

void main ( int argc, char *argv[] ) {
   char * cmds [ MAXCMDS ] ;
   FILE * fp_cmds, * fp_in, * fp_out ;

   progname = argv [0] ;
   if ( argc != 4 )
      error ( "Uso : %s cmds entrada salida", progname ) ;
   if ( ( fp_cmds = fopen ( argv[1], "r" )) == NULL )
      error ( "Imposible abrir archivo de comandos %s", argv[1] ) ;
   if ( ( fp_in = fopen ( argv[2], "r" )) == NULL )
```

```
      error ( "Imposible abrir archivo de entrada  %s", argv[2] ) ;
   if ( ( fp_out = fopen ( argv[3], "w" )) == NULL )
      error ( "Imposible abrir archivo de salida  %s", argv[3] ) ;

   read_cmds ( cmds, MAXCMDS, fp_cmds ) ;
   run_cmds  ( cmds, fp_in, fp_out ) ;
}

/*************************************************************
 *  Lee el archivo de Comandos...                            *
 *************************************************************/
void read_cmds ( char *cmds[], int limit, FILE *fp_cmds ) {
   int nl = 0 ;
   char line [MAXLINE] ;

   while ( fgets ( line, MAXLINE, fp_cmds ) ) {
      cmds [nl] = strdup ( line ) ;
      if ( !cmds [nl] )
        error ( "strdup falló en la linea \"%s\"", line ) ;
      if ( ++nl >= limit )
        error ( "read_cmds : demasiados comandos...", "" ) ;
   }
   cmds [ nl ] = NULL ;
}

/*****************************************
 * Aplica los comandos al archivo de entrada...          *
 *****************************************/
void run_cmds ( char *cmds[], FILE *fp_in, FILE *fp_out ) {
   char line [MAXLINE] ;

   while ( fgets ( line, MAXLINE, fp_in ) ) {
      apply_cmds ( cmds, line ) ;
      fputs ( line, fp_out ) ;
   }
}
```

```
/*****************************************************
 * Aplica los comandos a una linea del archivo de entrada...        *
 *****************************************************/
void apply_cmds ( char *cmds[], char *line ) {
   int i ;
   char * p ;

   for ( i = 0; p = cmds[i]; i++ ) {
      switch ( *p ) {
         case 's' :
           substitute ( ++p, line ) ;
           break ;
         default :
           error ( "Comando desconocido : %s", p ) ;
           break ;
      }
   }
}

/*************************************
 * Avisa de error y aborta la operación...        *
 *************************************/
void error ( char *s1, char *s2 ) {
   fprintf ( stderr, "%s\n", progname ) ;
   fprintf ( stderr, s1, s2 ) ;
   exit ( 1 ) ;
}

/*********************************************
 * Realiza un cambio de comando en una linea...        *
 *********************************************/
void substitute ( char *s, char *line ) {
   char *p ;
   int i, slen ;
   char delimiter = *s ;
   char        source [MAXLINE],
        target [MAXLINE],
        tmpstr [MAXLINE] ;

   for ( i = 0, s++; *s != '\n' && *s != delimiter; i++ )
     source [i] = *s++ ;
   source [i] = '\0' ;
   if ( ( slen = strlen (source) ) <= 0 )
     error ("Linea %s, el fuente no puede estar vacio", s ) ;
   for ( i = 0, s++; *s != '\n' && *s != delimiter; i++ )
```

```
    target [i] = *s++ ;
  target [i] = '\0' ;
  for ( p = strstr ( line, source ); p; p = strstr ( p + 1, source ) ) {
    strcpy ( tmpstr, p+slen ) ;
    strcpy ( p, target ) ;
    p += strlen (target ) ;
    strcpy ( p, tmpstr ) ;
  }
}
```

**El archivo de comandos que se utilizó como primer parámetro de ARRANGE para el desarrollo del controlador se muestra a continuación :**

```
s/DGROUP   group _DATA,_BSS/DGROUP     group _DATA,_BSS,_TEXT/
s/assume   cs:_TEXT/assume  cs:DGROUP/
s/dw  @/dw DGROUP:@/
s/offset       _/offset       DGROUP:_/
s/offset       @/offset       DGROUP:@/
```

# Apéndice C. Código fuente de la utilería CREADISK.

El objetivo de la utilería CREADISK fue detallado en el capítulo que describe la implementación del proyecto. A continuación se muestra su código fuente :

```
/***********************************************
 * Programa: CreaDisk                          *
 * Objetivo : Crea una imágen de disco (disco virtual)  *
 *          Para uso del controlador de discos virtuales  *
 ***********************************************/

#include <stdio.h>
#include <dos.h>
#include <dir.h>

#define BYTE    unsigned char
#define WORD    unsigned int
#define DWORD   unsigned long

/* Prototipos de las funciones utilizadas */
BYTE  creadisk      ( BYTE * sNombre, float fTamano ) ;
float EspacioEnDisco ( BYTE drv, BYTE Tipo ) ;

void main ( argc) {
 float Espacio ;

 char buffer [ MAXPATH ] ;
 char nombre [ MAXFILE ] ;
 char ext    [ MAXEXT  ] ;

 int rdo, cont ;
 float tamano ;

 printf ( "\nCreaDisk v1.00.Espejel.940720\n" ) ;
 printf ( "Utileria para crear imágenes de disco de tamaño arbitrario.\n" ) ;
 printf ( "Nota : El archivo se crea siempre en la raiz del disco C:.\n") ;
```

```
 if ( _argc != 3 ) {
   printf ( "\a\a\nError en el formato de la llamada. Sintáxis correcta :\n" ) ;
   printf ( "    CreaDisk  <NombreDisco>  <Tamaño>\n" ) ;
   printf ( "    NombreDisco : Nombre válido de archivo sin una ruta.\n" ) ;
   printf ( "    Tamaño      : Número real indicando el tamaño en MegaBytes.\n" ) ;
   exit ( 1 ) ;
 }

 /* Valida el nombre del archivo...*/
 rdo = fnsplit ( _argv [1], buffer, buffer, nombre, ext ) ;
 if( rdo & FILENAME ) {
  strcpy ( buffer, "C:\\") ;
  strcat ( buffer, nombre) ;
  if( rdo & EXTENSION ) strcat ( buffer, ext) ;
 }
 else {
       printf ( "\a\a\nError en el parámetro nombre de archivo. Programa
abortado...\n") ;
  exit ( 1 ) ;
 }

 /* Valida el parámetro tamaño... */
 for ( cont = 0; isdigit (_argv[2][cont]) || _argv[2][cont] =='.'; cont ++) ;
 if ( _argv[2][cont] != '\0' ) {
  printf ( "\a\a\nError en Parámetro tamaño. Programa abortado...\n") ;
  exit ( 1 ) ;
 }
 sscanf ( _argv [2], "%f", &tamano ) ;

 /* Valida el espacio disponible en el disco C:... */
 printf ( "\nTamaño de C: %3.2f MB's.\n", EspacioEnDisco('c','t') ) ;
 Espacio = EspacioEnDisco ( 'c', 'd' ) ;
 printf ( "Espacio disponible en C: %3.2f MB's.\n", Espacio ) ;
 if ( tamano > Espacio ) {
  printf ( "\a\a\nError : Espacio insuficiente en el disco. Programa abortado...\n") ;
  exit ( 1 ) ;
 }

 if ( creadisk ( buffer, tamano ) ) {
  printf ( "\a\a\nError : Imposible abrir archivo. Programa abortado...\n") ;
  exit ( 1 ) ;
 }
```

```
  printf ( "Proceso terminado.\n"
           "Nombre del nuevo archivo : %s. Tamaño : %4.2f MB's. \n\nGracias...\n",
           buffer, tamano) ;
}

/*************************************************************/
/* Declaración de estructuras y constantes usadas para     */
/* describir e inicializar el sector de Boot de un disco   */
/* virtual.                                                  */
/*************************************************************/

typedef struct bpb_struct_struct {
   WORD    bps ;
   BYTE    spCl ;
   WORD    res_sectores ;
   BYTE    num_FATs ;
   WORD    archs_raiz ;
   WORD    tamano_vol ;
   BYTE    media_byte ;
   WORD    spFAT ;
   WORD    spt ;
   WORD    hpc ;
   DWORD   ocultos ;
        DWORD   tamano_vol_32 ;
        BYTE  Drive_Fisico ;
        BYTE    Reservado ;
        BYTE  Signature ;
        DWORD       NumSerie ;
        BYTE  Etiqueta [11] ;
        BYTE  Fat12 [8] ;
} bpb_t ;

/*************************************
 * Estructura que define al Boot DOS                *
 *************************************/
typedef struct  BOOT_struct {
   BYTE    entry_point [3];   /* Salto al código de arranque    */
   BYTE    oem [8];           /* Nombre y Versión del fabricante */
   bpb_t   bpb;               /* Caracteristicas que definen al Disco */
} BootRecord;
```

```
/*************************************
 * Constantes usadas en el boot del disco...      *
 *************************************/
BYTE entry_point [3]  = { 0xEB, 0x3C, 0x90, '\0' } ;
BYTE oem        [9]  = "MSDOS5.0" ;
BYTE numserie   [5]  = { 0xFF, 0xFF, 0xFF, 0xFF, '\0' } ;
BYTE etiqueta   [11] = "E.Espejel." ;
BYTE fat12      [8]  = "FAT1 " ;
BYTE inicioFAT  [5]  = { 0xF8, 0xFF, 0xFF, '\0' } ;

/*************************************************
 * Rutina :  Creadisk.                            *
 * Objetivo : Crea un disco virtual de tamaño "fTamano"  *
 *          en el archivo especificado...         *
 * Parms :   sNombre : Nombre del archivo a crear *
 *          fTamano : Tamaño del disco virtual    *
 *************************************************/

BYTE creadisk ( BYTE * sNombre, float fTamano ) {

  FILE * fSalida;
  struct ffblk ffblk;
  int rdo ;
  BYTE buffer [512] ;
  BootRecord  * SectorArranque ;

  WORD Ts, Tc, Tf ;
  BYTE TipoFat ;

  int cont1, cont2 ;

  unsigned int nCursorDefault ;

  if ( fTamano <= 0 ) return 1 ;
  if ( EspacioEnDisco ( 'c', 'd' ) <= fTamano ) return 2 ;
  if ( ! findfirst( sNombre, &ffblk, 0 ) ) return 3 ;
  if ( ( fSalida = fopen ( sNombre, "wb" ) ) == NULL )  return 3 ;

  Ts = 2 * fTamano * 1024 ;
  Tc = Ts / 2 ;
  TipoFat = ( Tc >= 4086 ) ? 16 : 12 ;
  Tf = Tc / ( ( TipoFat == 16 ) ? 256 : 341 ) ;
  Tf ++ ;
```

```
for ( cont1=0; cont1 < 512; buffer [cont1++]='\0' ) ;

SectorArranque = ( BootRecord * ) buffer ;
strcpy ( SectorArranque->entry_point, entry_point ) ;
strcpy ( SectorArranque->oem, oem ) ;
SectorArranque->bpb.bps  = 512 ;
SectorArranque->bpb.spCl = 2 ;
SectorArranque->bpb.res_sectores = 1 ;
SectorArranque->bpb.num_FATs = 2 ;
SectorArranque->bpb.archs_raiz = 224 ;
SectorArranque->bpb.tamano_vol = ( Ts > 4096 ) ? 0 : Ts ; /* Calculado... */
SectorArranque->bpb.media_byte = 0xf8 ;
SectorArranque->bpb.spFAT = Tf ;          /* Calculado... */
SectorArranque->bpb.spt = 18 ;
SectorArranque->bpb.hpc = 8 ;
SectorArranque->bpb.ocultos = 0 ;
SectorArranque->bpb.tamano_vol_32 = (Ts >=4096) ? Ts:0; ; /* Calculado... */
SectorArranque->bpb.Drive_Fisico = 0 ;
SectorArranque->bpb.Reservado = 0 ;
SectorArranque->bpb.Signature = 0 ;
strcpy ( ( BYTE * ) & ( SectorArranque->bpb.NumSerie ), numserie ) ;
strcpy ( SectorArranque->bpb.Etiqueta, etiqueta ) ;
strcpy ( SectorArranque->bpb.Fat12, fat12 ) ;
SectorArranque->bpb.Fat12[4] = (TipoFat==12)? '2' : '6' ;

if ( fwrite ( buffer, 1, 512, fSalida ) != 512 ) return 3 ;
printf ( "Sector de Arranque terminado...\n" ) ;

for ( cont1=0; cont1 < sizeof(BootRecord) + 5; buffer [cont1++]='\0' ) ;

for ( cont1 = 1; cont1 <= 2; cont1 ++ ) {
  strcpy ( buffer, inicioFAT ) ;
  if ( TipoFat == 16 ) buffer [3] = 0xFF ;
  fwrite ( buffer, 1, 512, fSalida ) ;
  buffer [0] = buffer [1] = buffer [2] = buffer [3] = '\0' ;
  for ( cont2=2; cont2 <= Tf; cont2++) fwrite ( buffer, 1, 512, fSalida ) ;
}
printf ( "Areas de FAT terminadas...\n" ) ;

for ( cont1 = 1; cont1 <= 7; cont1 ++ ) {
  fwrite ( buffer, 1, 512, fSalida ) ;
}
printf ( "Directorio raiz terminado...\n" ) ;

printf ( "Area de datos :\n" ) ;
```

```
 for ( cont1=0; cont1 < 512; buffer [cont1++]= 0xF6 ) ;
 _AH = 0x03;
 _BH = 0x00 ;
 geninterrupt(0x10);       /* Obten el cursor default... */
 nCursorDefault = _CX ;
 _AH = 0x01;
 _CX = 0x2000;
 geninterrupt(0x10);       /* Apaga el cursor... */
 for ( cont1 = 1; cont1 <= Ts - 2 * Tf - 8; cont1 ++) {
   fwrite ( buffer, 1, 512, fSalida ) ;
   printf ( "%2.2f\% terminado...\r", cont1*100.0/(Ts-2.0*Tf-8) ) ;
 }
 printf ( "Area de datos terminada...\n" ) ;
 _AH = 0x01;
 _CX = nCursorDefault ;
 geninterrupt(0x10);

 fclose ( fSalida ) ;

 return 0 ;
}

typedef struct drvinfo {
 DWORD spc ;
 DWORD avail ;
 DWORD fatseg ;
 DWORD fatoff ;
 DWORD secsize ;
 DWORD clusters ;
 BYTE  fatid ;
} drive ;

/*****************************************
 * Rutina :   EspacioEnDisco.                    *
 * Objetivo : Invoca la interrupción 21h función 36h   *
 *          para averiguar el espacio total o disponible  *
 *          en un disco fisico.                  *
 * Parms.:    drv : Letra del drive.                *
 *          Tipo: 't' : retorna el espacio total del drive    *
 *                'd' : retorna el espacio libre del drive    *
 *****************************************/
float EspacioEnDisco ( BYTE drv, BYTE Tipo ) {
  union REGS regs ;
  struct SREGS segs ;
  int dn ;
```

```
  drive info ;

  drv = toupper (drv) ;
  dn = drv - 'A' + 1 ;

  regs.h.ah = 0x36 ;
  regs.h.dl = dn ;
  intdosx ( &regs, &regs, &segs ) ;
  info.spc = regs.x.ax ;
  info.avail = regs.x. bx ;
  info.secsize = regs.x.cx ;
  info.clusters = regs.x.dx ;

  if ( Tipo == 't' )
    return ( info.clusters*info.spc*info.secsize/1024.0/1024 ) ;
  if ( Tipo == 'd' )
    return ( info.avail*info.spc*info.secsize/1024.0/1024 ) ;
  else return -1 ;

}
```

# Apéndice D. Estructuras de un disco duro: A)Física genérica y B)Lógica en DOS.

## Introducción.

Una de las partes clave del código de la aplicación mostrada en este trabajo es la que se refiere al diseño e implementación del acceso a disco duro, ya que debido a la naturaleza de los controladores de dispositivos se debe realizar completamente a nivel BIOS. Ésto implica que no se cuenta con el soporte de la capa de software de DOS que facilita realizar operaciones de Entrada/Salida a disco (Sin importar si éste es flexible o duro). De hecho, no es posible ni siquiera accesar el Sistema de Archivos DOS desde BIOS; En otras palabras, desde BIOS el disco no se distingue como una partición con un grupo de directorios y archivos de tamaños y posiciones bien definidos sino como un medio magnético del cual se pueden leer y en el cual se pueden escribir bytes en una dirección específica, determinada por 3 parámetros que indican un sitio dentro de la estructura física del disco. A consecuencia de éste y debido a que en nuestro proyecto los archivos que simulan discos se encuentran en el sistema de archivos DOS, se necesita implementar código que interprete la información del disco duro formateado bajo DOS para llegar a los archivos que necesitamos.
Para lograr ésto se requiere un entendimiento claro de la forma en que un disco duro está organizado, desde los puntos de vista físico y lógico. El objetivo de este apéndice es proporcionar una guía para el entendimiento más claro de ambos aspectos.

## Estructura Física genérica de un disco duro.

Los discos para computadora son dispositivos basados en un "plato" rotatorio con superficies magnéticamente alterables, las cuales almacenan información digital. Para leer o escribir información sobre las superficies del disco se utilizan cabezas de lectura/escritura que pueden estar interconstruidas en la unidad de disco o bien ser independientes de ésta. Debido a que las cabezas de entrada/salida se pueden posesionar indiferentemente sobre cualquier posición en las superficies del disco, se dice que los discos son dispositivos de acceso aleatorio.
Se conoce como discos flexibles aquellos que están construidos con materiales maleables recubiertos de polvo magnético y que no tienen las cabezas de lectura/escritura interconstruidas. Comúnmente miden 3.5 o 5.25 pulgadas de diámetro y se escribe información sobre ambas superficies del disco. Este tipo de discos se introduce en una unidad de discos flexibles en la computadora.
Los discos duros siempre tienen las cabezas de lectura/escritura interconstruidas. Están constituidos por uno o más "platos" montados sobre una barra giratoria cuya velocidad de giro está gobernada por un pequeño motor. A

**cada superficie de cada "plato" en el disco se le asigna una cabeza de lectura/escritura. A su vez, todas las cabezas están montadas sobre brazos unidos que forman algo parecido a un "peine" el cual es controlado por un pequeño motor de pasos, el cual posee la habilidad de detener las cabezas de Lectura/Escritura en posiciones bien definidas (pasos) sobre la superficie del disco.**

**La siguiente figura ilustra estos conceptos.**

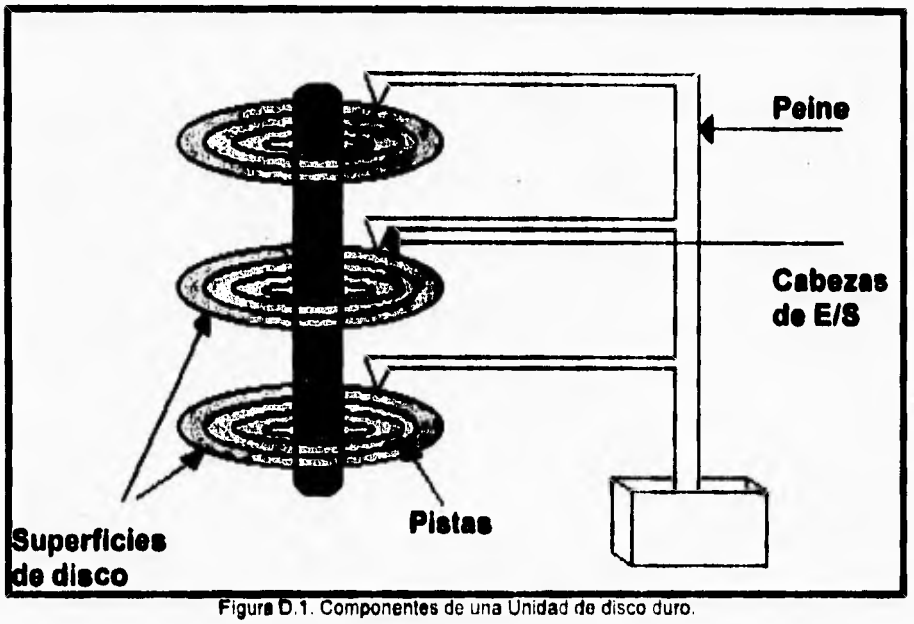



Figura D.1. Componentes de una Unidad de disco duro.

**A partir de este momento, se suceden una serie de capas de interfaz que permiten al usuario final utilizar un disco duro. Para empezar, Cualquiera que sea el tipo de disco que se use la unidad siempre es manejada por una tarjeta controladora de discos que se encuentra unida (o es parte de ella) a la tarjeta madre de la computadora. Dicha tarjeta emite las señales eléctricas requeridas para controlar las cabezas de lectura/escritura de la unidad de disco, proporcionando así la interfaz de bajo nivel para la lectura/escritura de información a discos.**
**La unidad controladora, a su vez, es manipulada desde software o firmware mediante envío de señales a los puertos de Entrada/Salida que tiene asignados.**
**La siguiente capa de interfaz, la que maneja dichos puertos, es proporcionada por las interrupciones BIOS de acceso a disco (La más importante de ellas es la 19, o13 hexadecimal). Estas permiten hacer operaciones de disco proporcionando**

como parámetros: Cabeza de Entrada/Salida, Pista y Sector. El significado de dichos parámetros se explicará en detalle más adelante.

Finalmente se encuentra la capa de interrupciones DOS, la cual es usada por la mayoría de las aplicaciones dirigidas al usuario final. Dicha capa facilita el esquema de peticiones pues requiere solo un parámetro: el sector lógico. Otras rutinas de estas interrupciones contienen código que permite accesar el Sistema de archivos DOS, la cual es la parte que se requiere implementar en nuestro proyecto.

Los siguientes términos ayudan a formalizar la descripción de la estructura de un disco duro.

## Cabeza.

La cabeza contiene un electromagneto posesionado sobre un soporte capaz de deslizarse sobre una de las superficies del disco duro. Polarizando positivamente o negativamente el electromagneto la cabeza pueda leer o escribir información de sobre la superficie del disco.

## Pista.

Como ya se mencionó antes, las cabezas de Lectura/Escritura de un disco están gobernadas por la acción de un motor de pasos. Cada paso que da dicho motor define un punto de descanso de la cabeza de Lectura/Escritura, momento en el cual ésta puede realizar transacciones de información al disco. Uno de éstos puntos de descanso, con la superficie del disco girando constantemente, define un círculo sobre ella. Evidentemente en una superficie de disco existen tantas pistas como pasos tenga el motor que controle a las cabezas de Escritura/Lectura.

## Sector.

Los sistemas de disco duro dividen cada Pista en arcos cortos (valores característicos podrían ser 17, 26 o 34 por cada Pista). Cada uno de estos arcos se conoce como Sector, y usualmente cada Sector contiene 512 bytes de datos.

## Cilindro.

Como ya se dijo, una Pista está definida por un círculo sobre una superficie de disco. A su vez, un disco puede estar formado por más de una superficie válida para leer o escribir. A la combinación de todas las pistas correspondientes a una misma posición del motor de pasos sobre las superficies del disco se le llama Cilindro.

**En la siguiente figura se muestran estos conceptos.**

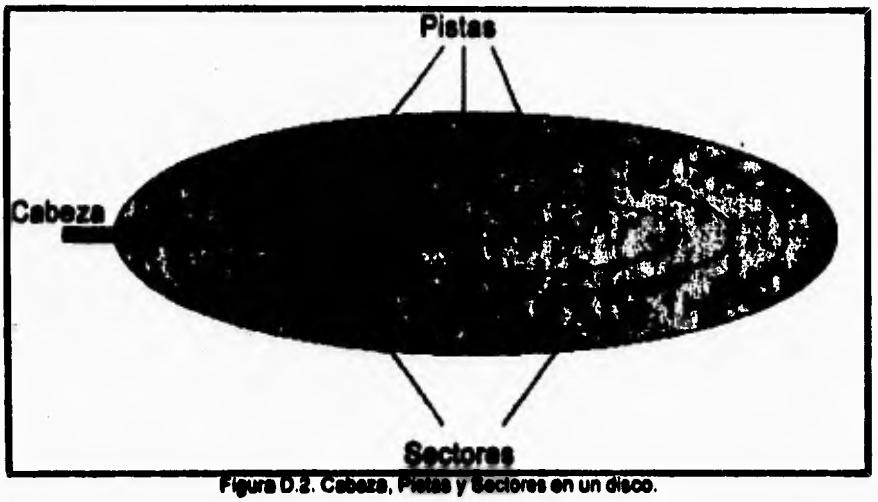



Figura D.2. Cabeza, Pistas y Sectores en un disco.

**A continuación se describirá a detalle la interfaz BIOS-Disco centrándonos en el uso de discos duros.**

**Como ya se hizo notar antes, el BIOS de una computadora PC incluye llamadas para control de discos flexibles y duros. El código de dichas llamadas programa directamente a la tarjeta controladora de las unidades de disco. De esta forma, en realidad el BIOS nunca escribe directamente del procesador hacia la unidad de disco del sistema. En algunas ocasiones datos y comandos son enviados directamente hacia la tarjeta controladora desde el microprocesador pero, más frecuentemente, el BIOS utiliza el controlador de DMA (Direct Memory Acces, Acceso directo a memoria). La ventaja de hacer uso de DMA es que BIOS (o cualquier otro código que use esta técnica) puede mover datos directamente desde y hacia la memoria del sistema sin intervención alguna por parte del microprocesador. Más específicamente, el controlador de discos utiliza el canal 2 de DMA para lograr esta comunicación. La relación entre las partes mencionadas se ilustra en la siguiente figura:**

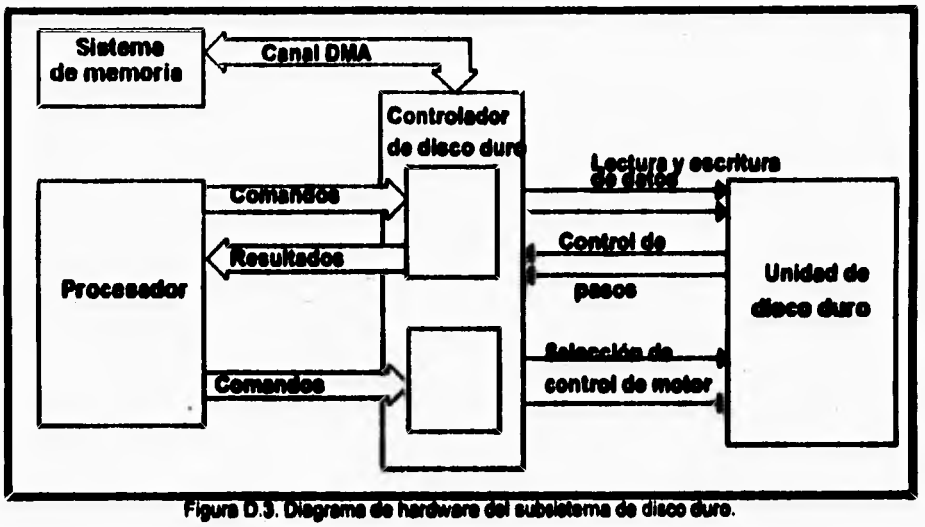



Figura D.3. Diagrama de hardware del subsistema de disco duro.

**Otras de las características físicas de importancia en un sistema que contenga al menos una unidad de disco es la siguiente: Para comunicarse con la unidad de disco y transmitirle instrucciones y datos, la tarjeta controladora utiliza una interfaz estándar bien determinada. las tres interfaces estándar mejor conocidas son:**

1. **ST506/ST412** **La interfaz estándar.**
2. **ESDI** **Enhanced small device interface, interfaz extendida para dispositivos pequeños.**
3. **SCSI** **Small Computer Systems Interface, interfaz para equipos de cómputo pequeños.**

**En la interfaz ST506 los pulsos generados por la unidad de disco son pasados directamente a la tarjeta controladora de discos usando uno de varios métodos de codificación de datos (el uso de uno u otro puede afectar significativamente el rendimiento de la unidad de disco). Los métodos más frecuentemente utilizados por esta interfaz son:**

1. **MFM** **Modified Frequency Modulation, Modulación por frecuencia modificada.**
2. **RLL** **Run Length Limited, Corrida de longitud limitada.**
3. **ARLL** **Advanced Run Length Limited, Corrida avanzada de longitud limitada.**

El conocimiento de estos parámetros es determinante para la adecuada comunicación entre la tarjeta controladora y las unidades de disco. Sin embargo, El BIOS no necesita preocuparse por estas caracteristicas. En vez de ésto, el BIOS utiliza los parámetros de unidad de disco duro almacenados en la tabla de tipos de unidades de disco que se encuentra en el área de datos de la ROM (Read Only Memory, Memoria de Solo Lectura). Normalmente dicha tabla se encuentra en la dirección F000:E401 Hexadecimal (en formato Segmento:Offset), aunque una llamada a la interrupción 13h función 8 (Read Drive Parameters, Lea parámetros de unidad). El formato de un renglón de dicha tabla es el siguiente:

| Offset | Tamaño | Descripción |
|---|---|---|
| 00h | 2 bytes | Número máximo de Cilindros |
| 02h | 1 byte | Número máximo de Cabezas |
| 03h | 2 bytes | Reservado |
| 05h | 2 bytes | Cilindro Inicial de Precompensation de Escritura |
| 07h | 1 byte | Reservado |
| 08h | 1 byte | Byte de control |
| 09h | 3 bytes | Reservado |
| 0Ch | 2 bytes | Cilindro para zona de aterrizaje de cabezas |
| 0Eh | 1 byte | Número de sectores por pista |
| 0Fh | 1 byte | Reservado |

Tabla D.1. Formato de la tabla BIOS de tipos de disco duro.

La tabla completa de las unidades de disco que un equipo puede soportar comúnmente difiere de un proveedor de chips ROM BIOS a otro. La tabla que se muestra en el apéndice dedicado a la interfaz de BIOS para manejo de disco duro muestra las opciones de declaración de disco duro que el BIOS de Phoenix Technologies incluya, por tanto no se consideró necesario repetirla en este apéndice.

La cantidad y tipo de unidades de disco que son soportadas por una computadora PC ha cambiado a través del tiempo. De tal forma, las tarjetas controladoras de disco duro para computadoras PX/XT (con procesadores 8086 u 8088) soportaban solo cuatro tipos diferentes de unidades de disco duro. En los sistemas PC/AT (con procesadores 80286) se pone a disposición del usuario final compatibilidad para 47 tipos de unidades de disco duro de uso común. En sistemas con procesadores más actualizados es común que aparte del soporte a estas 47 unidades se agregue una facilidad para editar un tipo de disco especifico en base a los parámetros comunes de un disco: Superficies (o cabezas), Pistas por superficie y Sectores por pista. La información acerca del nuevo tipo de unidad definida es almacenada en memoria RAM CMOS. Los servicios BIOS para acceso a disco duro soportan solo hasta 2 unidades de disco duro. Por otro lado, cuando se quiere agregar una interfaz ESDI se requiere reemplazar parte de los datos del BIOS que se encuentra en ROM.

A continuación se pasará a lo relativo a la distribución lógica de información a través de un disco duro, en particular si está formateado en DOS.

## Estructura Lógica de un disco duro en DOS.

Prácticamente Todo disco duro para computadora PC, sin importar el Sistema Operativo que tenga instalado, tiene siempre una parte fundamental: La tabla de Partición (El uso de la palabra *Prácticamente* obedece a que no existe ninguna regla escrita indicando que un Sistema Operativo debe respetar la existencia de dicha tabla, sin embargo el no hacerlo significaría el cerrarse a la posibilidad de convivencia con otros ambientes operativos). Dicha tabla proporciona un método mediante el cual un Sistema Operativo puede convivir con otros dentro de un mismo medio magnético. De esta forma, en una partición puede existir un Sistema de archivos UNIX mientras que en otra se tiene un Sistema de archivos DOS. Cualquiera que sea el Sistema operativo que exista en una partición determinada, éste debe proveer un medio para que el usuario pueda afectar la Tabla de partición y así indicar que la computadora deberá arrancar usando ahora otro ambiente operativo. Por lo común dicho medio de "Activación" de particiones está en forma de una utilería del Sistema operativo.
Cuando el Sistema operativo es DOS, existe al menos un ejemplar de las siguientes áreas dentro de un disco duro:

- Tabla de Partición.
- Registro de Arranque (Boot Record).
- Tabla de Registro de archivos (FAT, File Allocation Table).
- Directorio raíz (Root directory).
- Área de datos (Data area).

### Tabla de Partición.

Casi todo disco duro posee un Registro Maestro (Master Record), el cual reside en el primer sector físico del dispositivo ( el primer sector físico está ubicado en el cilindro -o pista- 0, cabeza -o lado- 0 y sector 1). Este registro maestro contiene código que es responsable de la lectura y descifrado de la Tabla de Partición contenida al final del mismo. El control pasa entonces al Registro de arranque (Boot Record) de la partición del disco duro que en ese momento esté señalada como activa (basado en la información contenida en la Tabla). La Tabla de Partición es una estructura que describe la forma en que el espacio de un disco está distribuido. Su tamaño es de 64 bytes y permite declarar hasta 4 particiones para un solo disco duro. Ésto implica que para cada partición se destinan 16 bytes. al final de la tabla se dejan 2 bytes que deberán tener el valor hexadecimal AA55h para indicar que la información contenida en la tabla es válida. En la siguiente tabla se muestra la estructura de cada uno de los renglones de la tabla de partición, acompañandose con valores que ejemplifiquen un renglón típico:

| Offset | Tamaño | Valor ejemplo | Descripción |
|---|---|---|---|
| 00h | 1 | 80h | Indicador de Arranque (Boot):<br>00h. No arranca<br>80h. Si arranca |
| 01h | 1 | 01h | Cabeza inicial |
| 02h | 1 | 01h | Sector inicial (Bits 0 al 5; los bits 6 y 7 son los 8 y 9 para cilindro inicial) |
| 03h | 1 | 00h | Cilindro inicial (Los 8 bits bajos) |
| 04h | 1 | 04h | Identificador del sistema:<br>3. Desconocido<br>4. DOS con FAT de 12 bits<br>5. DOS con FAT de 16 bits<br>6. DOS disco extendido con FAT de 16 bits |
| 05h | 1 | 04h | Cabeza final |
| 06h | 1 | 51h (11h) | Sector final (Bits 0 al 5; los bits 6 y 7 son los 8 y 9 para cilindro final) |
| 07h | 1 | E9h (1E9h) | Cilindro final (Los 8 bits bajos) |
| 08h | 4 | 00000011h | Primer sector de la partición |
| 0ch | 4 | 0000A2A1h | Sectores dentro de la partición |

Tabla D.2. Formato de un renglón de la Tabla de Partición.

**La información contenida en cada renglón de la Tabla de Partición indica en su mayoría parámetros que describen los límites físicos de la partición dentro del disco, pero hay 2 valores en particular interesantes, el indicador de arranque y el identificador de sistema. El indicador de arranque señala si dicha partición será la que deberá usarse para iniciar la computadora. En consecuencia, solo una (o ninguna) de las particiones deberá estar marcada como "arrancable" (La existencia de más de una provocaría conflictos durante la inicialización). La variable identificador de sistema es usada para indicar el tipo de Sistema operativo que está "montado" en la partición. Los valores que se muestran en la tabla anterior no son todos los que dicha variable puede tomar, pues otros Sistemas Operativos (como XENIX, UNIX o Pick) pueden extender la lista de posibles valores. Obviamente el formato descrito para la Tabla de partición no debe cambiar aunque se cambie de versión o marca de Sistema operativo, pues ésto provocaría serias dificultades de interoperabilidad entre Entornos (lo cual es precisamente el objetivo del concepto de Tabla de partición). Durante la inicialización del Sistema, El BIOS consulta el primer sector del disco para continuar con el proceso de arranque. En el caso de un disco flexible, dicho sector es el Sector de arranque. En un disco duro, se trata del Registro maestro ya descrito. Como parte del proceso BIOS localiza la Tabla de Partición y mediante el campo Indicador de Arranque determina cual Partición es la que deberá usarse para iniciar el sistema. Con dicha información se localiza el Sector de arranque de la partición activa y el proceso de inicialización continua como en el caso de un disco flexible. A partir de la versión 4.0 del sistema operativo MS-**

DOS se remueve la limitación de 32 MegaBytes por cada partición que dicho sistema operativo tenía en versiones anteriores.

## Registro o Sector de Arranque (Boot Record).

Una vez que el sistema ha determinado donde se encuentra el Registro de arranque (como ya dijimos en un disco flexible se encuentra en el primer sector mientras que en un disco duro su posición está determinada por la Tabla de Partición), BIOS lo carga en memoria. Típicamente, un Registro de arranque comienza con una instrucción de tres bytes de tamaño que, leída como código ensamblador, equivale a un salto a la rutina cargadora de arranque. Por ejemplo, en DOS versión 2.0 dicha instrucción equivale a los siguientes tres dígitos hexadecimales: "E9 8D 00", en DOS versión 3.2 a "EB 34 90", en DOS 4.01 es igual a "EB 3C 90", en DOS versión 5.0 a "EB 3C 90", etcétera. BIOS ejecuta el código del Registro de Arranque, lo cual lo hace saltar hacia la dirección indicada por los bytes dos y tres. En ese momento la rutina cargadora de arranque se ejecuta ocupándose de cargar el resto del Sistema Operativo.

La instrucción inicial de 3 bytes ya descrita es seguida por un campo de "Nombre de Sistema" de 8 bytes de longitud. Dicho campo identifica al fabricante cuyo sistema se encargó de formatear al disco.(Algunos fabricantes no escriben nada en este campo). Después viene el bloque de parámetros BIOS, BPB, cuya función es contener parámetros que describan la estructura física y lógica del disco en el cual está grabado. El formato de dicho bloque fue creciendo para soportar características que reflejan el avance tecnológico en técnicas del software o del hardware de un equipo. Por ejemplo, se agregaron previsiones para soportar discos de mayor tamaño, etiquetas de volúmen o número de serie de disco. La tabla siguiente muestra la estructura del BPB y sus cambios hasta la versión 4.0 de DOS.

| Corrimiento | Longitud (Bytes) | Significado |
|---|---|---|
| 00h | 2 | Número de bytes por sector. |
| 02h | 1 | Número de Sectores por Grupo (Tamaño de Cluster). |
| 03h | 2 | Número de sectores reservados. |
| 05h | 1 | Número de Tablas de almacenamiento de archivos (Número de FAT's). |
| 06h | 2 | Número máximo de entradas del directorio raíz. |
| 08h | 2 | Número total de sectores (o cero a partir de la versión 3.0 de DOS si es mayor a 65,535). |
| 0Ah | 1 | Descriptor de la unidad. |

| | | |
|---|---|---|
| 0Bh | 2 | Número de sectores por cada Tabla de almacenamiento de archivos (Tamaño de FAT). |
| 0Dh | 2 | Número de sectores por pista. |
| 0Fh | 2 | Número de cabezas. |
| 11h | 4 | Número de sectores ocultos. |
| 15h | 11 | Reservado (Antes de la versión 3.0). |

Tabla D.3. Formato del Bloque de Parámetros BIOS de un disco DOS.

**Las extensiones realizadas en la versión 3.0 son las siguientes:**

| Corrimiento | Longitud (Bytes) | Significado |
|---|---|---|
| 15h | 4 | Número total de sector si la palabra en el corrimiento 08h tiene valor de cero. |
| 19h | 7 | Reservado. |

Tabla D.4. Extensiones al Bloque de Parámetros BIOS en DOS versión 3.0

**Las extensiones correspondientes a la versión 4.0 se indican a continuación:**

| Corrimiento | Longitud (Bytes) | Significado |
|---|---|---|
| 19h | 1 | Número físico de unidad. |
| 1Ah | 1 | Reservado. |
| 1Bh | 1 | Byte de identificación para registro de arranque extendido. |
| 1Ch | 4 | Número de serie del volúmen (creado a partir de la fecha y hora). |
| 20h | 11 | Etiqueta del volúmen. |
| 2Bh | 8 | Reservado. |

Tabla D.5. Extensiones al Bloque de Parámetros BIOS en DOS versión 4.0

**Note particularmente como las diferentes actualizaciones al Sistema Operativo han intentado, en cada cambio, mantener la compatibilidad con versiones anteriores, aun yendo al extremo de proveer dos diferentes campos para el número total de sectores de datos cuando fue posible (gracias al avance de la tecnología de manejo de información en medios magnéticos) tener más de 65,535 sectores (inicialmente se reservaron solo 2 bytes para almacenar este valor) en un solo disco. Desafortunadamente, no todos los fabricantes de sistemas DOS proveyeron dejando áreas reservadas para futuro crecimiento. Ésto puede provocar que discos formateados con dichas versiones del Sistema Operativo sean ilegibles cuando otras versiones sean instaladas.**

**En todas las versiones de DOS el BPB es critico a la operación del programa de arranque pues debe conocer estos parámetros para poder encontrar y cargar el BIOS y el kernel. Ésto se debe a que hasta antes de la versión 3.0 los programas cargadores asumían que la versión del BPB que se encontraba en el BIOS de la memoria ROM era aplicable al momento de arrancar el sistema. Por tanto, nunca hacían uso de los datos que se encontraban en el sector de arranque. Como resultado, algunas firmas (entre ellas Tandy y Heath-Zenith) omitían los datos BPB de los discos flexibles formateados bajo DOS versión 2.0. Obviamente ésto provocaba sistemas más cerrados puesto que se suponía el tipo de disco A priori. Si un nuevo formato de disco hubiera aparecido el sistema habría sido incapaz de leerlo. Cuando DOS versión 3.0 apareció, estos discos se volvieron ilegibles pues este Sistema Operativo trataba de leer los parámetros descriptores del disco donde realmente se encontraba el código de arranque. Debido a que el Sistema Operativo IBM DOS versión 2.0 produjo discos que se apegaban a la escritura de un BPB en los discos, éstos fueron aceptados en versiones posteriores de DOS. Éste creó la convicción generalizada de que el nuevo Sistema Operativo buscaba las iniciales "IBM" en los bytes 3,4 y 5 del registro de arranque y si las encontraba daba al disco un tratamiento diferente. Una situación similar se dio con el salto de la versión 3.0 a la 4.0 del Sistema Operativo IBM DOS. Sin embargo, en este caso IBM DOS v4.0 realmente buscaba las iniciales IBM. Si no las encontraba simplemente marcaba un "Error de medio desconocido". Ésta estrategia tomada por IBM fue reprobada por los consumidores pues hizo a IBM DOS v4.0 un sistema más cerrado y debido a ésto perdió terreno en la aceptación del público. Hechos como éste marcan la pérdida de supremacía de IBM en el campo del cómputo de escritorio y el inicio de preeminencia de Microsoft. De cualquier forma, con solo cambiar la cadena en las direcciones de la 3 a la 10 del registro de arranque de un disco a "IBM V2.0" o "IBM V3.0" se logra que sea aceptado por el sistema, sin perder aceptación por parte de los demás Sistemas DOS.**

**Aunque el área destinada al Registro de arranque es usualmente de solo un sector de tamaño, nada impide que su tamaño sea mayor es por ésto que también se ha designado con el nombre de "Área de sectores reservados".**
**El siguiente dibujo ilustra la estructura de Registro de arranque dentro de un disco duro, además de mostrar de forma somera las demás áreas lógicas creadas por el Sistema Operativo DOS.**

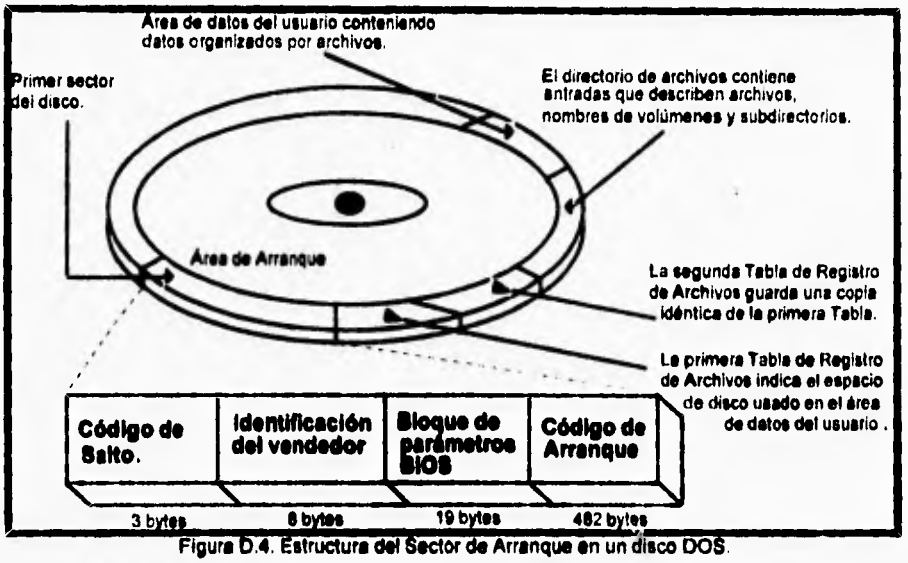



Figura D.4. Estructura del Sector de Arranque en un disco DOS.

## Tabla de Registro de archivos. (FAT, File Allocation Table).

**El Sistema Operativo DOS utiliza una estructura denominada Tabla de Registro de archivos (mejor conocida en el medio de computación por sus siglas en inglés como FAT, File Allocation Table) para administrar el área de datos de un disco. Esta Tabla indica a DOS cuales porciones del disco están destinadas a almacenar un archivo. También divide al disco en estructuras recursivas de organización de archivos denominadas Directorios. Debido al carácter crítico de las funciones que esta estructura realiza, comúnmente un disco en formato DOS contiene más de una copia de ella. Por lo normal son dos las copias que se mantienen, aunque nada determina que tenga que ser así. De cualquier forma, todas las copias que existan están almacenadas una detrás de otra en el disco. Conforme la situación de un archivo se modifica (En su tamaño, posición, existencia, fecha u hora de última modificación o distribución) DOS refleja su nueva situación en la Tabla principal, pero también detecta la existencia y posición de las copias de respaldo que existan y las afecta para que reflejen consistentemente la información contenida en la Tabla principal. Quien se encarga de indicar a DOS la cantidad, tamaño y tipo de Tablas de Registro de archivos que existen es precisamente el Registro de Arranque (Boot record), el cual en su estructura BPB contiene esta información.**

Es interesante notar que hasta la varaión 6.0 de MS-DOS (cuando se incorporan utilerías adquiridas por tarceros al sistema, como SCANDISK) ningún comando propio de DOS posee la capacidad de localizar y aprovechar las copias de respaldo de la Tabla de Registro de archivos. Si la primera Tabla se dañaba se debía recurrir a utilerías independientes (al principio ni siquiera las proveían los dos principales proveedores de Sistemas DOS: IBM y Microsoft) para resolver el problema. Sin embargo, también debe resaltarse que en la práctica casi cualquier desastre que afecte a la primera Tabla terminará por afectar también a las subsecuentes (si éstas existen). Éste hace muy cuestionable la utilidad de mantener copias de respaldo para las Tablas de Registro de archivos, aunque es innegable que en ciertos casos pueden resultar determinantes para la exitosa recuperación de información valiosa en un disco.
Cada Tabla de Registro consiste en una serie de entradas, en algunas ocasiones de 12 y en otras de 16 bits de largo cada una, que registran el estado de cada Cluster (Grupo de sectores) dentro de un disco. Cuando se utilizan entradas de 12 bits, dos de ellas se empacan dentro de 3 bytes consecutivos dentro de la Tabla. Éste optimiza el uso del espacio dentro de cada Tabla. A continuación se explicará el concepto de Cluster.

## Grupo de sectores (Cluster).

El administrar un disco asignando archivos a sectores puede resultar un proceso muy poco eficiente. Si, por ejemplo, tomamos un disco de 10 MegaBytes de capacidad, contendría aproximadamente 20,000 sectores y mantener una entrada para cada uno de ellos dentro de la Tabla de Registro de archivos crearía una Tabla muy grande. Si la Tabla usara entradas de 12 bits entonces una copia de ésta mediría 30,000 sectores o sea 15 KiloBytes, y si existieran 2 copias de la Tabla en el disco entonces se estaría destinando un total de 30 KiloBytes. Aunque seguramente este espacio se puede destinar a almacenar más archivos, no parece ser demasiado con respecto a los 10 MegaBytes de tamaño del disco. Sin embargo, una Tabla de menor tamaño seguramente sería mucho más fácil de administrar por parte del Sistema Operativo, y más rápida de guardar y cargar. Se requeriría también menos espacio en memoria RAM, la cual es mucho más valiosa que el espacio en disco. Una solución a éste es el agrupar sectores consecutivos, de tal forma que cuando se cree un nuevo archivo se amplíca por asignarle una cierta cantidad de sectores contiguos dentro del disco (no un solo sector) para su almacenamiento, y marca la Tabla de Registro de archivos en la entrada correspondiente (así como las copias que existan). Este concepto es el de Grupo de sectores (Cluster). Otra forma de llamar a estos grupos es como "Unidades de almacenamiento", puesto que conforman la unidad básica de almacenamiento de archivos en un disco formatsado bajo el Sistema Operativo DOS.
Los sectores que componen un Grupo no necesariamente han de estar en la misma superficie o pista; el primer sector de la primera superficie de la primera pista podría ser el primero en conformar el primer Grupo, y de ahí los Grupos se seguirían secuencialmente.

El número de sectores en un Grupo es siempre una potencia de 2, pues de esta forma se simplifica la conversión entre número de Grupo y sus números de sectores lógicos correspondientes. En la siguiente tabla se muestran algunos tamaños de Grupo para ciertos tipos de discos.

| Tipo de disco | Grupo de sectores |
|---|---|
| Disco flexible 3½ pulgadas doble densidad | 2 |
| Disco flexible 5¼ pulgadas baja densidad | 1 |
| Disco flexible 5¼ pulgadas doble densidad | 2 |
| Disco duro de 10MB | 8 |
| Disco duro (AT) 20MB | 4 |

Tabla D.6. Tamaños de *clusters* típicos.

En la actualidad los discos flexibles comúnmente tienen un tamaño de Grupo de 2 sectores; los primeros discos duros usaban tamaño de Grupo de 8 sectores, pero los usuarios encontraron demasiado grande 4 KiloBytes cuando muchos archivos pequeños eran almacenados -Se perdía demasiado espacio. Con la introducción de la versión 3.0 del Sistema Operativo MS-DOS el tamaño de Grupo para discos duros grandes fue reducido a 4 sectores (2 KiloBytes). Para cada disco, el tamaño de Grupo es uno de los parámetros clave para la operación del sistema y se encuentra registrado en el BPB.

Dentro de la Tabla de Registro de archivos (FAT), cada entrada corresponde exactamente a un Grupo de sectores en el disco. La entrada correspondiente al Grupo 0 guarda el código del disco, y la del Grupo 1 es siempre llenada con bits en 1 (Para entradas de 16 bits FFFF hexadecimal, y para entradas de 12 bits FFF). El primer Grupo designado como disponible para almacenar datos es siempre el 2.
Cuando un Grupo en el disco está disponible para almacenamiento el valor de su entrada correspondiente en la Tabla de Registro es 0. Cuando un Grupo es asignado a un archivo, se almacena en su entrada de la Tabla de Registro el valor hexadecimal FFFF para 16 bits, y FFF para 12 bits. De esta forma, se indica que es el último Grupo que conforma a un archivo. A su vez, el número del Grupo inicial del archivo es registrado en un renglón del directorio correspondiente (Dicha estructura se describirá más adelante). Conforme el archivo crece y nuevos Grupos le son asignados, las marcas hexadecimales FFF/FFFF se van recorriendo para indicar siempre el último Grupo del archivo y el número del Grupo recién asignado es escrito en la entrada de Tabla que antes contenía el indicador de fin de archivo. De esta forma en la Tabla de Registro de archivos se forma una cadena (que, vista como una estructura de datos, equivale a una lista simplemente ligada) que indica cuales son los Grupos de sectores (*Clusters*) que componen a un archivo, sin importar que tan dispersos se encuentren éstos en el disco.
La siguiente ilustración muestra las relaciones recién descritas:

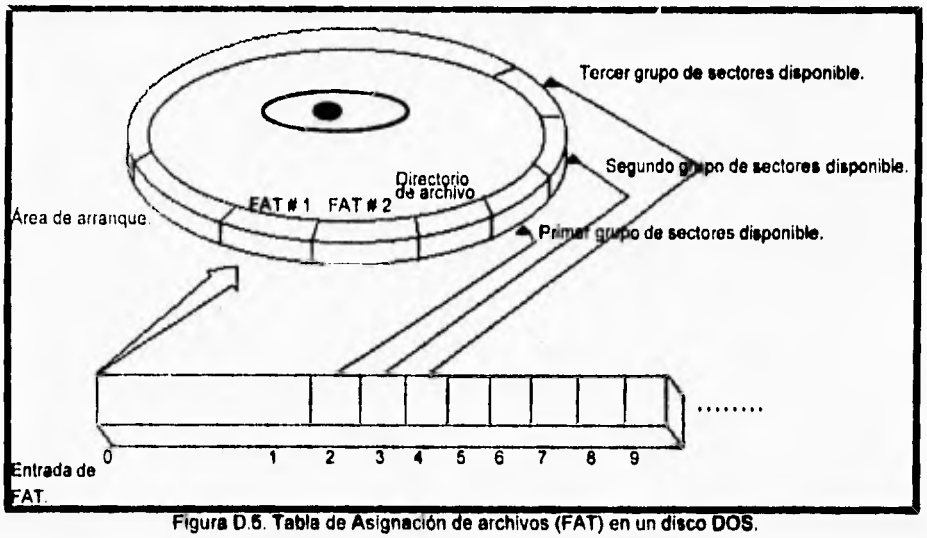



Figura D.6. Tabla de Asignación de archivos (FAT) en un disco DOS.

**Se utilizan códigos especiales para indicar si un Grupo está dañado, y si es así, en que forma. Los valores hexadecimales FF7/FFF7 al FFE/FFFE son utilizados para éste propósito.**
**Como ya se vio, existen 2 tipos de Tablas de Registro de archivos: de 12 y de 16 bits. A continuación se describe cada uno de ellos.**

## Tabla de Registro de archivos (FAT) de 12 bits.

**Una tabla de Registro de 12 bits resulta 25% más pequeña que una de 16 bits. Dada la siguiente Tabla de Registro de 12 bits se realizará su análisis:**

| FD | FF | FF | 03 | 40 | 00 | 05 | 60 | 00 | 07 | 80..... |
|---|---|---|---|---|---|---|---|---|---|---|

**De la descripción antes realizada se desprende que las dos primeras entradas de la Tabla (los 3 primeros bytes) contienen información del sistema. Por tanto, los Grupos 0 y 1 del área de datos no son accesibles para almacenar información. Los siguientes 12 bits indican el valor del Grupo 2, etcétera. Las fórmulas que nos permiten averiguar los valores de dos entradas contiguas a partir de los 3 Bytes que las componen son las siguientes (Todos los números son hexadecimales):**

$$Entrada_1 = ( (Byte_2 \text{ AND } 0F) * 1000 ) + Byte_1$$

**$Entrada_2$ = ( $Byte_3$ * 10 ) + ( ( $Byte_2$ AND F0 ) / 10 )**

**Aplicando estas fórmulas en el ejemplo mostrado para obtener los valores de los Grupos 2 y 3 se obtiene lo siguiente:**

**$Entrada_2$ = ( ( 40 AND 0F ) * 1000 ) + 03**
**= ( ( 0 ) * 1000 ) + 03**
**= 3**
**$Entrada_3$ = ( 00 * 10 ) + ( ( 40 AND F0 ) / 10 )**
**= 0 + ( 40 / 10 )**
**= 4**

**Según estos resultados, las entradas 2 y 3 no son FFF ni tampoco indican Grupos dañados. Por tanto, cada una de ellas apunta a la siguiente entrada del Grupo que conforma un archivo. Como la entrada 2 apunta a la 3 y la entrada 3 apunta a la cuatro, los Grupos 2, 3 y 4 forman parte de un solo archivo (No sabemos si el archivo comienza en el Grupo 2). Como ya se vio, el valor en una entrada de la Tabla de Registro puede implicar varias cosas. La siguiente tabla muestra sus diferentes significados:**

| Categoría | Código |
|---|---|
| Libre, para asignación | 0 |
| Parte de un archivo (apuntador al siguiente grupo) | 2 al FF6 |
| Grupo dañado | FF7 |
| Fin de la cadena de grupo | FF8-FFF |

Tabla D.7. Significado de una celda para una *FAT* de 12 bits.

## Tabla de Registro de archivos (FAT) de 16 bits.

**La liberación de MS-DOS versión 3.0 trajo consigo soporte para discos duros de mayor tamaño y una Tabla de Registro de archivos de 16 bits por cada entrada. El siguiente texto representa el inicio de una Tabla de Registro de archivos (FAT) de 16 bits típica :**

**F8 FF FF FF 03 00 04 00 05 00 06.....**

**La traducción en este caso es mucho más sencilla que para el caso de la Tabla de 12 bits. Las primeras dos entradas (4 Bytes) son usados para información del sistema. Cada entrada subsecuente dentro de la Tabla ocupa dos bytes. Los 2 bytes de la entrada para el Grupo 2 indican un valor de 3, lo cual implica que el archivo continua en el Grupo 3, y así. Al igual que en la Tabla de 12 bits, cada**

entrada puede significar diferentes cosas. La siguiente tabla muestra el significado de los posibles valores de una entrada.

| Categoría | Código |
|---|---|
| Libre, para asignación | 0 |
| Parte de un archivo (apuntador al siguiente grupo) | 2 al FFF6 |
| Grupo dañado | FFF7 |
| Fin de la cadena de grupo | FFF8-FFFF |

Tabla D.8. Significado de una celda para una *FAT* de 16 bits.

La Tabla de Registro de archivos reserva, pero no hace uso, del espacio para las entradas 0 y 1. El primer Byte de la Tabla es usado como identificador de disco, y ayuda a definir su formato. A continuación se muestran sus probables valores:

| Valor | Características del disco |
|---|---|
| F0h | No identificable |
| F8 h | Disco duro |
| F9h | Doble densidad, 15 sectores/pista |
| F9h | Doble densidad, 9 sectores/pista (720 KiloBytes) |
| FCh | Baja densidad, 9 sectores/pista |
| FDh | Doble densidad, 9 sectores/pista (360 KiloBytes) |
| FEh | Baja densidad, 8 sectores/pista |
| FFh | Doble densidad, 8 sectores/pista |

Tabla D.9. Significado de la primera celda de una *FAT*.

Después de la última copia de la Tabla de Registro se encuentra el área destinada al Directorio Raiz (Root Directory), el cual se detallará a continuación.

## Directorio Raíz (Root Directory).

Los primeros Sistemas Operativos para microcomputadoras ()CP/M, TRSDOS, Apple DOS, etcétera) trataban a los archivos de un disco en la misma forma; todos los archivos se encontraban en un solo directorio. DOS hacia lo mismo hasta antes de que se liberara la versión 2.0. Empezando con esta versión un concepto de organización de archivos fue tomado de Sistemas como UNIX y XENIX. Este esquema de directorios jerárquicos permite una fácil clasificación y manipulación de cantidades grandes de archivos. En un Sistema con directorios jerárquicos cada disco tiene un único Directorio Raiz (Root Directory) de tamaño predefinido, ubicado en una posición bien determinada del disco. Este Directorio raíz parece un directorio plano como el de cualquier disco de la versión 1.0 de MS-DOS, pero tiene una ventaja: En el se pueden definir Subdirectorios, los cuales son semejantes a los archivos comunes, salvo por una diferencia: su contenido es a su vez un directorio, en el cual se pueden incluir más archivos y definir a su vez más Subdirectorios. De hecho, el nivel límite de anidación para la definición

de Subdirectorios es de 32 niveles (Esta restricción está impuesta por el tamaño límite de 65 caracteres de una cadena descriptora de un archivo, suponiendo que se encuentre en directorios de una sola letra).
Como ya se dijo antes, el tamaño y posición del Directorio Raíz en el disco están predefinidos durante la operación del sistema y sus valores están registrados en el BPB del Registro de arranque (Boot Sector). El tamaño del directorio raíz se mida en archivos. Es decir, el BPB indica cual es el número máximo de archivos o directorios que se pueden crear en el Directorio raíz. Por lo tanto, se puede decir que un Directorio es un arreglo de estructuras que representan archivos, donde cada estructura representa una entrada del Directorio. Es por tanto importante saber como está formada cada una de estas estructuras si queremos comprender a un Directorio.
Cada entrada de Directorio tiene un tamaño de 32 bytes y contiene información descriptiva acerca de un archivo o directorio, como Nombre, Extensión, atributos, tamaño, posición de inicio dentro del disco, y fecha y hora de su última modificación. La siguiente tabla muestra la estructura básica de una entrada de Directorio:

| Offset | Tamaño | Significado |
|---|---|---|
| 00h | 8 bytes | Nombre del archivo |
| 08h | 3 bytes | Extensión del archivo |
| 0Bh | Byte | Atributos del archivo |
| 0Ch | 10 bytes | Reservado (no usar) |
| 16h | Palabra | Hora de la última actualización |
| 18h | Palabra | Fecha de la última actualización |
| 1Ah | Palabra | Grupo de sectores inicial |
| 1Ch | Doble palabra | Tamaño del archivo |

Tabla D.10. Formato de una entrada de Directorio.

A continuación se explicará cada uno de los campos mostrados.

## Nombre del archivo.

Los primeros 8 Bytes de la entrada de Directorio indican el Nombre con que el usuario conoce al archivo o directorio. Esta es la explicación de que los Nombres de archivos en DOS estén limitados a 8 Bytes. Si un nombre de archivo es más corto que 8 bytes aparecerá justificado a la izquierda y completado con Espacios hasta completar 8 caracteres. Además, DOS ajusta todos los nombres para que sean mayúsculas.
El primer byte de este campo, dependiendo de su valor, tiene varios significados especiales. Éstos se muestran en la siguiente tabla:

| Primer byte | Significado |
|---|---|
| 00h | La entrada de directorio nunca ha sido usada; ninguna sigue. |
| 05h | El primer carácter de un nombre de archivo es E5h |

| | |
|---|---|
| 2Eh | La entrada es un alias para un subdirectorio. Si el siguiente byte es 2Eh, el directorio apunta comenzando en el campo del disco conteniendo el nombre del grupo o del directorio padre actual |
| E5h | Archivo borrado |

Tabla D.11. Significado del primer byte de una Entrada de directorio.

La explicación de cada uno es la siguiente:

- El código 0 ahorra a DOS búsquedas infructuosas de más entradas de Directorio. Cuando este código es encontrado durante la búsqueda de un archivo es interpretado como "Fin de Directorio" y la búsqueda concluye.
- El carácter E5 hexadecimal puede ser utilizado como el primer carácter de un Nombre de archivo pero es almacenado en la entrada de Directorio como el carácter ASCII 5. El motivo para realizar esta traducción queda explicado leyendo el siguiente inciso.
- Un E5 hexadecimal como el primer carácter marca archivos que han sido borrados; DOS ignora dichas entradas cuando busca archivos o las reusa cuando nuevos archivos son creados. Ésto provee un mecanismo mediante el cual se puede recuperar un archivo que haya sido borrado por equivocación.
- La entrada 2E hexadecimal es un punto (carácter '.'). Se encuentra solo en Subdirectorios y marca una entrada de directorio como un Subdirectorio que apunta al Subdirectorio actual. Si el siguiente Byte es también un punto, la entrada de Directorio es un Subdirectorio que apunta al Directorio padre del Directorio actual. De tal forma, se pueden advertir siempre las entradas de nombre "." y ".." en todo Subdirectorio de un disco formateado en DOS; Son creadas automáticamente en cuanto el Directorio se crea y no pueden ser removidas sino hasta que el Directorio en que están se ha borrado. Por ejemplo, si se enviara a DOS el comandos "DEL .", éste lo interpretará como "DEL *.*", lo cual probablemente no sea el efecto deseado.

## Extensión.

Los tres caracteres que siguen al Nombre del archivo se conocen como Extensión. Las mismas convenciones que aplican al Nombre del archivo son válidas para este campo.

## Atributos.

El objetivo de este campo es doble: Primero, definir el tipo de la entrada de Directorio (Subdirectorio o archivo) y Segundo, definir sus características de accesibilidad. Cada bit de este byte marca una característica particular de la entrada de Directorio.

La siguiente tabla muestra el uso de cada bit dentro del campo.

| Bit 76543210 | Significado |
|---|---|

| | |
|---|---|
| .....X | Solo lectura |
| .....X. | Oculto |
| .....X.. | Sistema |
| .....X... | Etiqueta de volúmen |
| .....X.... | Subdirectorio |
| .....X..... | Archivo |
| XX..... | Reservado |

Tabla D.12. Campo de Atributo de una Entrada de directorio.

La explicación para cada uno es la siguiente:

- Solo lectura. Si este bit está en 1, el archivo no puede ser borrado ni se puede escribir en si.
- Oculto. Una entrada marcada con este bit no puede ser vista con los comandos de DOS. Sin embargo, éste no afecta su comportamiento en ningún otro aspecto.
- Sistema. Marca comúnmente aplicada a los archivos propietarios del Sistema operativo, como IO.SYS; Sin embargo, el usuario los pueden aplicar a los propios también. Su uso previene a éstos archivos de ser borrados involuntariamente.
- Etiqueta de volúmen. Si el bit está prendido la entrada de Directorio es la Etiqueta (el nombre) del disco.
- Subdirectorio. Indica que la entrada contiene un Subdirectorio.
- Archivo. Este bit se prende cuando se ha hecho una actualización a un archivo. Típicamente, programas de realización de respaldos en DOS utilizan este campo para saber que archivos han sido modificados y por tanto deben respaldarse.

## Hora de la última actualización.

Este campo es inicializado consultando el reloj del sistema cuando se crea el archivo, y actualizado posteriormente cuando un archivo que se haya modificado es cerrado.

La tabla siguiente muestra el significado de cada bit en este campo.

| °Bits FEDCBA98 | Significado 76543210 | |
|---|---|---|
| XXXXX.. | . . . . . | Hora |
| .....XXX | XXX . . . . | Minutos |
| ..... | . . . XXXXX | Incremento de 2 segundos |

Tabla D.13. Formato del campo *Hora* de una Entrada de directorio.

La hora está contenida en 5 bits (formato 24 horas) y los minutos en 6. Debido a que ésto deja solo 5 bits para los segundos, éstos son divididos en 2. Debido a

ésto la aproximación de la hora solo llega hasta el número par de segundos más cercano.

### Fecha de la última actualización.

Este campo es muy semejante al anteriormente descrito en cuanto a sus caracteristicas de inicialización y actualización. La siguiente tabla muestra el significado de cada bit dentro de los dos bytes que conforman el campo.

| Bits FEDCBA98 | Significado 76543210 | |
|---|---|---|
| XXXXXXX . | ..... | Contador de años (relativo a 1980) |
| ..... X | XXX ..... | Meses |
| ..... | ... XXXXX | Días |

Tabla D.14. Formato del campo *Fecha* de una Entrada de directorio.

El año es almacenado en 7 bits, el mes en 4 y el día en 5.
Se debe notar que la entrada de año es relativa a 1980. En otras palabras, es un desplazamiento a partir de la fecha base de 1980, no un valor absoluto. Por ejemplo, el año 1995 está almacenado como 15. El haber dejado 7 bits para este campo implica que el máximo año que se puede representar es 2107.

### Grupo de Sectores Inicial.

Es una palabra que indica cual es el Grupo de sectores (Cluster) inicial de un archivo. Con este valor, se revisa la entrada correspondiente dentro de la Tabla de Registro de archivos (FAT) para averiguar el Grupo siguiente, y así sucesivamente.

### Tamaño del archivo.

El campo de tamaño indica el tamaño exacto, en bytes, del archivo. De la longitud de este campo (4 bytes) se puede inferir que el tamaño máximo de un archivo en DOS es de 4,294'967,295 bytes, aproximadamente 4 GigaBytes. Como puede verse, este campo no requerirá modificación por algún tiempo.

### Área de Datos.

El Área de datos es la última área lógica que DOS define en un disco. Es donde se almacena el grueso de la información: los archivos del usuario. Comienza inmediatamente después de haber terminado el Directorio Raiz, y se extiende hasta el final del disco o hasta que la partición correspondiente termine.

# Apéndice E. Servicios BIOS de Disco duro.

## Introducción.

**Una parte de gran importancia para el proyecto propuesto es la relacionada con el código que se encarga de realizar la interfaz a los servicios que el BIOS de una PC proporciona para accesar un disco duro, los cuales pueden considerarse como bastante primitivos. Los servicios que permiten accesar información de disco organizada como archivos están programados por los desarrolladores del Sistema Operativo, Es decir, forman parte de una capa de software más alta y de más fácil uso que la de BIOS. Sin embargo, el código de un Controlador de dispositivos no puede permitirse hacer llamadas a dicha capa de software debido a la naturaleza No reentrante de DOS (Se considera que un Controlador de dispositivos es parte del Sistema operativo, por tanto llamar desde éste a funciones del Sistema operativo provocaría conflictos de Reentrancia).**

**De lo antes expuesto y del hecho que al Controlador de dispositivos que se pretende desarrollar en este trabajo requiere que se realice gran cantidad de accesos al sistema de archivos de un disco formateado en DOS, se entiende que es imprescindible contar con sólidos conocimientos acerca de la forma en que BIOS accesa un disco duro.**

**Este apéndice complementa la información proporcionada por el que describe las características físicas y lógicas de un disco y trata de proporcionar la información necesaria para comprender la interfaz de BIOS al disco.**
**A lo largo de este apéndice se hará uso extensivo de numeración hexadecimal. Cuando se utilicen números en esta base se usara la letra h para posfijar el número. Por ejemplo, 14h equivale a 20 en base decimal.**

**Los servicios de BIOS para disco duro permiten realizar lectura, escritura, formato, diagnóstico, inicialización y otras operaciones para hasta 2 unidades de disco duro*.**
**La forma de invocar los servicios BIOS de disco flexible y duro es a través de la interrupción 13h. Si una unidad de disco duro está presente, los servicios de disco duro redireccionan al vector de la interrupción 13h a la dirección 00:4Ch de la tabla de vectores de interrupción.**

* Actualmente comienza a tomar aceptación una interfaz a unidades de disco duro conocida en inglés como "Enhanced IDE", cuyo propósito es partir de la arquitectura IDE para proporcionar capacidades extendidas de disco. Algunas de estas extensiones son: Mayor velocidad de transferencia de información, Medios más compactos, mejor relación Economía/Desempeño con respecto a opciones como SCSI, y hasta cuatro dispositivos de almacenamiento por cada equipo. La desventaja de esta interfaz es que requiere que la computadora utilice otro BIOS, el cual redefine la interfaz de acceso de BIOS a Disco duro.

Por otro lado, cuando un Disco duro está presente BIOS redirecciona todas las peticiones de servicio a unidades de disco flexible a la interrupción 40h. Este redireccionamiento es transparente para el usuario final, el cual puede seguir invocando la interrupción 13h sin importar si solicita servicio a disco duro o a flexible.

Para tratar de mostrar la información relacionada a los servicios de disco duro de la forma más completa posible, la información presentada en este apéndice se ha clasificado en las siguientes secciones:

1. Listado de funciones de Servicio a disco duro.
2. Datos de memoria RAM del sistema relacionados.
3. Datos de memoria RAM de CMOS relacionados.
4. Datos del BIOS en memoria ROM relacionados.
5. Puertos de Entrada/Salida relacionados.
6. Manejo de errores realizado por la interrupción 13h.
7. Descripción de las funciones de Servicio a disco duro.

## 1. Listado de funciones de Servicio a disco duro.

Los servicios de BIOS a disco duro incluyen 23 funciones. La función a realizar se selecciona a través del registro AH. La siguiente tabla sumariza estas funciones:

| Función | Descripción | Tipo de controlador |
|---|---|---|
| 00h | Inicializa unidades de Disco Duro y Disco flexible | [XT] [AT] |
| 01h | Lee el estado del Disco Duro | [XT] [AT] |
| 02h | Lee Sectores | [XT] [AT] |
| 03h | Escribe Sectores | [XT] [AT] |
| 04h | Verifica Sectores | [XT] [AT] |
| 05h | Formatea cilindro | [XT] [AT] |
| 06h | Formatea Pista defectuosa | [XT] |
| 07h | Formatea Unidad | [XT] |
| 08h | Lee Parámetros de Unidad | [XT] [AT] |
| 09h | Inicializa parámetros de Unidad | [XT] [AT] |
| 0Ah | Lectura larga de sectores | [XT] [AT] |
| 0Bh | Escritura larga de sectores | [XT] [AT] |
| 0Ch | Busca Cilindro | [XT] [AT] |
| 0Dh | Inicialización Alterna del Disco Duro | [XT] [AT] |
| 0Eh | Diagnostico 1: Lee Buffer de prueba | [XT] |
| 0Fh | Diagnostico 2: Escribe Buffer de prueba | [XT] |
| 10h | Prueba si la unidad está lista | [XT] [AT] |
| 11h | Recalibra la Unidad | [XT] [AT] |
| 12h | Diagnóstico de memoria RAM del controlador | [XT] [AT] |

| | | |
|---|---|---|
| 13h | Diagnóstico de la unidad del controlador | [XT] [AT] |
| 14h | Diagnóstico interno controlador | [XT] [AT] |
| 15h | Lee tipo de Disco Duro | [XT] [AT] |

Tabla E.1. Funciones de Servicio BIOS a disco duro.

## 2. Datos de memoria RAM del sistema relacionados.

**El área de datos de BIOS (Direcciones 400h a la 500h de memoria RAM) contiene, entre otra información, alguna relacionada con los servicios de BIOS a disco duro. La siguiente tabla muestra estas zonas:**

| Localización | Tamaño | Descripción |
|---|---|---|
| 40:74h | 2 bytes | Estado de la última operación de disco duro, donde:<br><br>00h = No hubo error<br>01h = Solicitud de función inválida<br>02h = Marca de dirección no encontrada<br>03h = Error de protección contra escritura<br>04h = Sector no encontrado<br>05h = La Inicialización Falló<br>07h = Falla en la actividad del parámetro del Drive<br>08h = Exceso en operación de DMA<br>09h = Error en el límite de datos<br>0Ah = Bandera de sector dañado localizada<br>0Bh = Pista defectuosa localizada<br>0Dh = Número inválido de sectores en formateo<br>0Eh = Marca de dirección de datos detectada<br>0Fh = Nivel de arbitraje DMA fuera de rango<br>10h = Error de ECC o CRC incorregible<br>11h = Error de datos ECC corregido<br>20h = Falla general del Controlador<br>40h = Falla en operación de búsqueda<br>80h = Tiempo agotado<br>AAh = Unidad no lista<br>BBh = Un error indefinido ocurrió<br>CCh = Falla de escritura en la unidad seleccionada<br>E0h = Error de status/Registro de error es 0<br>FFh = Falla en operación del búsqueda |
| 40:75h | 1 byte | Número de unidades de disco duro (Solo AT) |
| 40:76h | 1 byte | Control de byte (cuando la escritura es un puerto de E/S 03F6h) |
| 40:77h | 1 byte | Offset del Puerto de disco duro |
| 40:8Ch | 1 byte | Status del Controlador de disco duro (Solo AT) |

| | | |
|---|---|---|
| 40:8Dh | 1 byte | Status de error del Controlador de disco duro (Solo AT) |
| 40:8Eh | 1 byte | Bandera de Interrupción de disco duro (Solo AT) |

Tabla E.2 Zonas de Memoria RAM del sistema relacionadas a los servicios BIOS a disco duro.

## 3. Datos de memoria RAM de CMOS relacionados.

**Los datos de memoria RAM de CMOS referenciados por estos servicios se muestran en la siguiente tabla:**

| CMOS RAM Offset (hex) | Tamaño | Descripción |
|---|---|---|
| 0Eh | 1 byte | Estado de Diagnostico , donde:<br>Bit 7 = 1 Reloj de Tiempo real perdió poder<br>Bit 6 = 1 CRC de memoria RAM CMOS erroneo<br>Bit 5 = 1 Configuración inválida al arrancar<br>Bit 4 = 1 Error de tamaño de memoria al arrancar<br>Bit 3 = 1 Disco duro o controlador falló su inicialización<br>Bit 2 = 1 Registro de hora inválido<br>Bit 6 = 1 Controlador no coincide en configuración<br>Bit 0 = 1 Tiempo agotado en lectura de controlador |
| 11h | 1 byte | Tipo de unidad de disco duro 0 |
| 12h | 1 byte | Tipo de unidad de disco duro 1 |
| 19h | 1 byte | Tipo de Control del disco duro 0 |
| 1Ah | 1 byte | Tipo de Control del disco duro 1 |

Tabla E.3. Zonas de Memoria RAM de CMOS relacionadas a los servicios BIOS a disco duro.

## 4. Datos del BIOS en memoria ROM relacionados.

**El código BIOS utiliza los parámetros de unidad de disco duro almacenados en la tabla de tipos de unidades de disco que se encuentra en el área de datos de la ROM (Read Only Memory, Memoria de Solo Lectura). Normalmente dicha tabla se encuentra en la dirección F000:E401 Hexadecimal (en formato Segmento: Offset), aunque una llamada a la interrupción 13h función 8 (Read Drive Parameters, Lee parámetros de unidad). El formato de un renglón de dicha tabla es el siguiente:**

| Offset | Tamaño | Descripción |
|---|---|---|
| 00h | 2 bytes | Número máximo de Cilindros |
| 02h | 1 byte | Número máximo de Cabezas |
| 03h | 2 bytes | Reservado |
| 05h | 2 bytes | Cilindro inicial de Precompensation de Escritura |
| 07h | 1 byte | Reservado |
| 08h | 1 byte | Byte de control |

| 09h | 3 bytes | Reservado |
|---|---|---|
| 0Ch | 2 bytes | Cilindro para zona de aterrizaje de cabezas |
| 0Eh | 1 byte | Número de sectores por pista |
| 0Fh | 1 byte | Reservado |

Tabla E.4. Formato de un renglón de la Tabla de Unidades de disco duro en ROM.

**La tabla completa de las unidades de disco que un equipo puede soportar comúnmente difiere de un proveedor de chips ROM BIOS a otro. La tabla que se muestra a continuación es la que el BIOS de Phoenix Technologies incluye. En este BIOS, las entradas 48 y 49 son editables por el usuario final y sus valores quedan almacenados en RAM CMOS.**

| # | Cilindros | Cabezas | Precompensación de Escritura | Zona de aterrizaje | Sectores |
|---|---|---|---|---|---|
| 1 | 306 | 4 | 128 | 305 | 17 |
| 2 | 615 | 4 | 300 | 615 | 17 |
| 3 | 615 | 6 | 300 | 615 | 17 |
| 4 | 940 | 8 | 512 | 940 | 17 |
| 5 | 940 | 6 | 512 | 940 | 17 |
| 6 | 615 | 4 | -1 | 615 | 17 |
| 7 | 462 | 8 | 256 | 511 | 17 |
| 8 | 733 | 5 | -1 | 733 | 17 |
| 9 | 900 | 15 | -1 | 901 | 17 |
| 10 | 820 | 3 | -1 | 820 | 17 |
| 11 | 855 | 5 | -1 | 855 | 17 |
| 12 | 855 | 7 | -1 | 855 | 17 |
| 13 | 306 | 8 | 128 | 319 | 17 |
| 14 | 733 | 7 | -1 | 733 | 17 |
| 15 | 612 | 4 | 0 | 633 | 17 |
| 17 | 977 | 5 | 300 | 977 | 17 |
| 18 | 977 | 7 | -1 | 977 | 17 |
| 19 | 1024 | 7 | 512 | 1023 | 17 |
| 20 | 733 | 5 | 300 | 732 | 17 |
| 21 | 733 | 7 | 300 | 732 | 17 |
| 22 | 733 | 5 | 300 | 733 | 17 |
| 23 | 306 | 4 | 0 | 336 | 17 |
| 24 | 1024 | 7 | -1 | 1024 | 17 |
| 25 | 615 | 4 | 0 | 615 | 17 |
| 26 | 1024 | 4 | -1 | 1024 | 17 |
| 27 | 1024 | 5 | -1 | 1024 | 17 |
| 28 | 1024 | 8 | -1 | 1024 | 17 |
| 29 | 512 | 8 | 256 | 512 | 17 |
| 30 | 615 | 2 | 615 | 615 | 17 |

| | | | | | |
|---|---|---|---|---|---|
| 31 | 989 | 5 | 0 | 989 | 17 |
| 32 | 1020 | 15 | -1 | 1024 | 17 |
| 35 | 1024 | 9 | 1024 | 1024 | 17/26 |
| 36 | 1024 | 5 | 512 | 1024 | 17 |
| 37 | 830 | 10 | -1 | 830 | 17 |
| 38 | 823 | 10 | 256 | 824 | 17 |
| 39 | 615 | 4 | 128 | 664 | 17 |
| 40 | 615 | 8 | 128 | 664 | 17 |
| 41 | 917 | 15 | -1 | 918 | 17 |
| 42 | 1023 | 15 | -1 | 1024 | 17 |
| 43 | 823 | 10 | 512 | 823 | 17 |
| 44 | 820 | 6 | -1 | 820 | 17 |
| 46 | 925 | 9 | -1 | 925 | 17 |
| 47 | 699 | 7 | 256 | 700 | 17 |
| 48 | | | | | |
| 49 | | | | | |

Tabla E.5. Tabla de Unidades de disco duro en ROM.

**En la tabla anterior:**

- **Los renglones 15, 33, 34 y 45 se consideran reservados.**
- **Los renglones 48 y 49 se dejan como editables por el usuario, y su valor se almacena en RAM CMOS.**

## 5. Puertos de Entrada/Salida relacionados.

**Los servicios de la interrupción 13h referencian los puertos de Entrada/Salida mostrados en la siguiente tabla:**

| Dirección E/S | Status E/S | Descripción |
|---|---|---|
| 0020h | L | Controlador del interruptor programable. Registros de Solicitud de interrupción/Servicio de Interrupción equivalentes al Registro de manejo de Interrupción, donde:<br>Bits 7-0 = 0 No hay Solicitud activa para la línea de interrupción correspondiente<br>= 1 Solicitud activa para la línea de interrupción correspondiente |
| 00020h | L | Registro indicador de Interrupción en servicio, donde:<br>Bits 7-0 = 0 La línea de interrupción correspondiente no está siendo servida actualmente<br>= 1 La línea de interrupción correspondiente está siendo servida actualmente |

| | | |
|---|---|---|
| 0020 | E | Controlador de interrupción Programable. Comando de inicialización palabra 1 (ICW1) (El bit 4 es 1), donde:<br>Bits 7-5 = 000 únicamente usados en modo 80/85<br>Bit 4 = 1 Reservado<br>Bit 3 = 0 Modo de rodear el disparador<br>= 1 Modo de nivel del disparador<br>Bit 2 = 0 Los vectores de interrupción sucesivos están separados por 8 bytes<br>= 1 Los vectores de interrupción sucesivos están separados por 8 bytes<br>Bit 1 = 0 Modo de cascada<br>= 1 Modo singular- no necesita el ICW3<br>Bit 0 = 0 No necesita el ICW4<br>= 1 Necesita el ICW4 |
| 0021h | E | ICW2, ICW3 o ICW4 en orden secuencial después de ICW1 escritos en el puerto 0020h<br><br>ICW2, donde:<br>Bits 7-3 = Líneas de Dirección A0-A3 para el controlador de interrupciones<br>Bits 2-0 = 000 Reservados<br><br>ICW3, donde<br>Bits 7-0 = 0 No hay un Controlador esclavo unido al pin de interrupción correspondiente<br>= 1 Hay un Controlador esclavo unido al pin de interrupción correspondiente<br><br>ICW4, donde:<br>Bits 7-5 = 000 Reservado<br>BIT 4 = 0 Modo anidado<br>Bits 3-2 = 00 Modo de no buffer<br>= 01 Modo de no buffer<br>= 10 Modo buffer/ esclavo<br>= 11 Modo buffer/ maestro<br>Bit 1 = 0 Normal EOI<br>= 1 Auto EOI<br>Bit 0 = 0 Modo 80/85<br>= 1 Modo 8086/8088 |

| | | |
|---|---|---|
| 0021h | L/E | Registro de máscara de interrupción (OCW1), donde:<br>Bit 7 = 0 Interrupción de impresión en paralelo habilitada<br>Bit 6 = 0 Interrupción de disco flexible habilitada<br>Bit 5 = 0 Interrupción de disco duro habilitada<br>Bit 4 = 0 Interrupción serial habilitada<br>Bit 3 = 0 Reservado<br>Bit 2 = 0 Interrupción de video habilitada<br>Bit 1 = 0 Interrupción de teclado/diapositivo apuntador / RTC habilitada<br>Bit 0 = 0 Interrupción de temporizador habilitada |
| 0021h | E | OCW2 (bit 4 es 0, bit 3 es 0), donde:<br>Bits 7-5 = 000 Rotación en modo automático EOI (limpiar)<br>= 001 EOI No especifico<br>= 010 No hay operación<br>= 011 EOI Especifico<br>= 100 Rotación en modo automático EOI (Set)<br>= 101 Rotación en comando EOI no especifico<br>= 110 Comando de prioridad<br>= 111 Rotación en comando EOI especifico<br>Bit 4 = 0 Reservado<br>Bit 3 = 0 Reservado<br>Bits 2-0 = Solicitud de interrupción a la que el comando aplica |
| 0020h | E | OCW3 (Bit 4 es 0, Bit 3 es 1),donde:<br>Bit 7 = 0 Reservado<br>Bits 6-5 = 00 No hay operación<br>= 01 No hay operación<br>= 10 Marcas especiales del Reset<br>= 11 Marcas especiales del Set<br>Bit 4 = 0 Reservado<br>Bit 3 = 1 Reservado<br>Bit 2 = 0 no hay Comando Poll<br>= 1 Comando Poll<br>Bits 1-0 = 00 No operación<br>= 01 No operación<br>= 10 Solicitud del registro de interruptor de lectura en la siguiente lectura del puerto 0020h<br>= 11 Interruptor de lectura en el registro de servicio de la lectura siguiente al puerto 0020h |

| | | |
|---|---|---|
| 0070h | E | Puerto para registro de dirección del RAM CMOS, donde:<br>Bit 7 = 1 NMI Deshabilitado<br>= 0 NMI Habilitado<br>Bits 6-0 = 0 Dirección CMOS |
| 0071h | L/E | Puerto de registro de Datos RAM CMOS |
| 00A0h | L/E | Controlador de interrupción programable 2 |
| 00A1h | E | Máscara del controlador de interrupción programable 2, donde:<br>Bit 7 = 0 Reservado<br>Bit 6 = 0 Interrupción de disco duro habilitada<br>Bit 5 = 0 Interrupción de Excepción 80387 habilitada<br>Bit 4 = 0 Interrupción del ratón habilitada<br>Bit 3 = 0 Reservado<br>Bit 2 = 0 Reservado<br>Bit 1 = 0 Redirección en cascada habilitada<br>Bit 0 = 0 Interrupción de reloj de tiempo real habilitada |
| 0170h | L/E | Registro de datos del disco duro 1 (único AT) |
| 0171h | L/E | Registro de error del disco duro 1 (único AT) |
| 0172h | L/E | Cuenta de sectores del disco duro 1 (único AT) |
| 0173h | L/E | Numero de sectores del disco duro 1 (único AT) |
| 0174h | L/E | Número Bajo de cilindros del disco duro 1 (único AT) |
| 0175h | L/E | Número Alto de cilindros del disco duro 1 (único AT) |
| 0176h | L/E | Registro de Unidad/Cabeza del disco duro 1 (único AT) |
| 0177h | L/E | Registro de Estatdo del disco duro 1 (único AT) |
| 01F0h | L/E | Registro de datos del disco duro 0 (único AT) |
| 01F1h | L/E | Registro de error del disco duro 0 (único AT) |
| 01F2h | L/E | Cuenta de sectores del disco duro 0 (único AT) |
| 01F3h | L/E | Numaro de sectores del disco duro 0 (único AT) |
| 01F4h | L/E | Número Bajo de cilindros del disco duro 0 (único AT) |
| 01F5h | L/E | Número Alto de cilindros del disco duro 0 (único AT) |
| 01F6h | L/E | Registro de Unidad/Cabeza del disco duro 0 (único AT) |
| 01F7h | L/E | Registro de Estatdo del disco duro 0 (único AT) |
| 0320h | L/E | Registro controlador de disco duro (8 o 16 bits) |

Tabla E.6. Puertos de Entrada/Salida relacionados a los Servicios BIOS a disco duro.

## 6. Manejo de errores realizado por la interrupción 13h.

Al retorno de cada función los servicios de disco duro indican el resultado de la operación con un código de error numérico. Este código de error es retornado en el registro AH y es almacenado en el byte de Estado de disco duro (en la dirección 40h:74h).

Si las funciones retornan exitosamente de cumplir con su objetivo guardan el valor 00h en el registro AH y ponen en 0 (apagan) el bit Carry Flag del registro

*Flags Register* del procesador. De lo contrario, AH contiene el código de error y el bit Carry Flag se prende.

Los posibles códigos de error retornados se muestran en la siguiente tabla:

| Código de error | Descripción |
|---|---|
| 00h | Sin error |
| 01h | Código de Función inválida pasado en AH o parámetro inválido |
| 02h | Marcas de dirección no encontradas |
| 04h | Sectores no encontrado |
| 05h | Fallas en Inicialización |
| 07h | Fallas en la actividad del parámetro de unidad |
| 08h | Exceso en la operación DMA |
| 09h | Error en el límite de datos |
| 0Ah | Bandera de sector defectuoso detectada |
| 0Bh | Cilindro defectuoso detectado |
| 0Dh | Número de sectores inválido en formateo |
| 0Eh | Marca de dirección de datos de Control detectada |
| 0Fh | Nivel de arbitraje DMA fuera de rango |
| 10h | Error ECC o CRC Incorregible |
| 11h | Error de datos ECC corregido |
| 20h | Falla de Controlador |
| 40h | Falla en búsqueda |
| 80h | Tiempo agotado |
| AAh | Unidad no está lista |
| BBh | Error Indefinido |
| CCh | Falla en escritura en la unidad seleccionada |
| E0h | Error de statua/ Registro de error es 0 |
| FFh | Falla en el sentido de operación |

Tabla E.7. Códigos de error retornados por las funciones de Servicio BIOS a disco duro.

## 7. Descripción de las funciones de Servicio a disco duro.

### Función 00h: Inicializa unidades de disco duro y disco flexible.

Esta función inicializa los controladores de las unidades de disco duro y de disco flexible. Además, recalibra las posiciones de las cabezas de Lectura/Escritura al cilindro 0.

El número de disco fijo (80h para el primer disco físico y 81h para el segundo disco físico) es especificado en el registro DL. Para inicializar solo la controladora se puede usar la función 0Dh.

| | | | |
|---|---|---|---|
| Entrada | AH | 00h | |
| Salida | AH | 00h | Sin errores. |
| | | XXh | Código del error |
| | | | Estado de disco fijo (Dirección 40h:74h). |
| | CF | 0 | Sin errores. |
| | | 0 | Hubo errores. |

Tabla E.8. Interfaz para la función 00h de Servicio BIOS a disco duro.

El estado de la última operación de disco flexible es almacenado en la dirección 40:41h, y el estado de la última operación de disco duro es almacenado en la dirección 40:74h.

## Función 01h: Lee el estado del disco duro.

Esta función retorna el código de resultado obtenido después de la última operación de disco en el registro AL. Antes de retornar de la llamada, esta función establece el estado de disco duro (El valor que se retornaría si esta función fuera nuevamente llamada) al valor 00h y limpia la bandera Carry Flag.

| | | | |
|---|---|---|---|
| Entrada | AH | 01h | |
| | DL | | Número de unidad de disco físico. |
| | | 80h | Disco duro 1. |
| | | 81h | Disco duro 2. |
| Salida | AH | 00h | Sin errores. |
| | | XXh | Código del error |
| | | | Estado de disco fijo (Dirección 40h:74h). |
| | AL | | Código de error de la ultima operación. |

Tabla E.9. Interfaz para la función 01h de Servicio BIOS a disco duro.

## Función 02h: Lee sectores.

Esta función lee el número de sectores especificado como parámetro en el registro AL de la unidad especificada en el registro DL a un área de buffer definida por el par de registros ES:BX (formato Segmento:Offset). También lee las 2 palabras del código de corrección de error (ECC, Error correction Code) asociado con ese sector y automáticamente corrige errores de ECC (Vease una explicación breve de esta técnica de corrección de errores en el Glosario). Las transferencias de varios sectores a la vez son terminadas después de que cualquier sector marque un error de lectura, en cuyo caso la función completa falla a pesar del número de sectores que si se hayan logrado leer. Los números de Cilindro, Sector y Cabeza son especificados en los registros CH, CL y DH respectivamente.

El número de sectores especificado en el registro AL debe ser diferente de cero y no debe exceder 128, pues números mayores a 128 causan una transferencia de

más de 64 KiloBytes y forzan de esa forma un error de límites del controlador de Acceso directo a memoria (DMA, Direct Access Memory).

El llamador de esta rutina debería reintentar esta función al menos tres veces, enviando la función de Servicio BIOS de Inicialización (Función 00h) entre cada llamada, cuando una condición de error ocurra para asegurar que no es solo un problema de temporización de hardware.

| Entrada | AH | 02h | |
|---|---|---|---|
| | AL | | Número de sectores a leer. |
| | CH | | Número del cilindro (los 8 bits bajos). |
| | CL | | Número de Cilindro/Sector, donde: |
| | | | Bits 7 y 6: Número del cilindro (los 2 bits altos). |
| | | | Bits 5 al 0: Número de sector. |
| | DH | | Número de cabeza. |
| | DL | | Número de unidad física: |
| | | | 80h: Disco duro 1. |
| | | | 81h: Disco duro 2. |
| | ES:BX | | Apuntador al buffer. |
| Salida | AH | 00h | Sin errores. |
| | | XXh | Código del error |
| | | | Estado de disco fijo (Dirección 40h:74h). |
| | AL | | Número de sectores de datos transferidos. |
| | CF | 0 | Sin errores. |
| | | 1 | Hubo error. |

Tabla E.10. Interfaz para la función 02h de Servicio BIOS a disco duro.

## Función 03h: Escribe sectores.

Esta función escribe el número de sectores especificado en AL a la unidad especificada en DL desde un área de buffer apuntada por los registros ES:BX (En formato Segmento:Offset). El número de cabeza es indicado en DH. Los números de cilindro y sector son indicados en CH y CL como se muestra posteriormente.
No se requiere una llamada previa a la función de búsqueda de cilindro.
El número de sectores especificado en AL no debe ser cero ni exceder 128, pues los números mayores a 128 causen una transferencia de más de 64 KiloBytes y se forza un error de límites de Acceso directo a memoria (DMA).
Si algún error ocurre cuando esta función es invocada, llame a la función 00h (Inicialización) y reintente al menos 2 veces.
La Tabla especificando la interfaz de llamada a esta función es la E.11.

## Función 04h: Verifica sectores.

Esta función verifica el número de sectores especificado en AL en el cilindro, cabeza y unidad especificados en CH, CL y DH respectivamente.

**Esta función no compara datos en disco con datos en memoria. Solo verifica que los sectores especificados puedan ser leidos y que el Chequeo de redundancia ciclica (CRC, Cyclical Redundancy Check) sea correcto. No se causa ninguna transferencia de datos de disco hacia memoria o viceversa.**
**La Tabla especificando la interfaz de llamada a esta función es la E.12.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **03h** | |
| | **AL** | | **Número de sectores a escribir.** |
| | **CH** | | **Número del cilindro (los 8 bits bajos).** |
| | **CL** | | **Número de Cilindro/Sector, donde:** |
| | | | **Bits 7 y 6: Número del cilindro (los 2 bits altos).** |
| | | | **Bits 5 al 0: Número de sector.** |
| | **DH** | | **Número de cabeza.** |
| | **DL** | | **Número de unidad fisica:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| | **ES:BX** | | **Apuntador al buffer.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **AL** | | **Número de sectores de datos transferidos.** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.11. Interfaz para la función 03h de Servicio BIOS a disco duro.

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **04h** | |
| | **AL** | | **Número de sectores a verificar.** |
| | **CH** | | **Número del cilindro (los 8 bits bajos).** |
| | **CL** | | **Número de Cilindro/Sector, donde:** |
| | | | **Bits 7 y 6: Número del cilindro (los 2 bits altos).** |
| | | | **Bits 5 al 0: Número de sector.** |
| | **DH** | | **Número de cabeza.** |
| | **DL** | | **Número de unidad fisica:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **AL** | | **Número de sectores de datos verificados.** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.12. Interfaz para la función 04h de Servicio BIOS a disco duro.

## Función 05h: Formatea cilindro.

**Esta función se usa para formatear el cilindro especificado en CH y CL usando la cabeza especificada en DL. Si el sistema utiliza un controlador de disco duro tipo XT la función 0Fh debe ser llamada previamente.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **05h** | |
| | **AL** | | **Factor de interleave (solo en controladoras tipo XT).** |
| | **CH** | | **Número del cilindro (los 8 bits bajos).** |
| | **CL** | | **Número de Cilindro/Sector, donde:** |
| | | | **Bits 7 y 6: Número del cilindro (los 2 bits altos).** |
| | | | **Bits 5 al 0: Número de sector.** |
| | **DH** | | **Número de cabeza.** |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| | **ES:BX** | | **Apuntador a la tabla de entradas de campos de 2 bytes.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.13. Interfaz para la función 05h de Servicio BIOS a disco duro.

**Contenidos de la tabla apuntada por ES:BX:**

**El programa que invoque a esta función debe proveer una tabla de 512 bytes de longitud conteniendo marcadores de dirección. Debe estar apuntada por ES:BX (formato Segmento:Offset). En la tabla debe haber una entrada de 2 bytes por cada sector en el cilindro. Las entradas de la tabla deben estar formateados como se muestra a continuación:**

**Byte 1:** **Bandera Bien/Mal:**
**00h: Bien**
**80h: Mal.**

**Byte 2:** **Número de sector.**

**Por ejemplo, el campo de dirección para formatear una pista en una unidad de disco duro que tiene un factor de interleave de 2 y que está formateado para 17 sectores por pista sería:**

```
db    00,01,00,0A,00,02,00,0B,00,03,00,0C,00,04,00,0D
db    00,05,00,0E,00,06,00,0F,00,07,00,10,00,08,00,11
db    00,09
```

**Si algún error ocurre al llamar esta función, llama a la función 00h para inicializar el controlador de disco duro y reintente esta función a menos tres veces.**

## Función 06h: Formatea pista defectuosa.

**Esta función inicializa una pista, escribiendo campos de dirección de disco duro y sectores de datos y prendiendo banderas de sectores defectuosos. Está diseñada para uso con un controlador de disco duro tipo XT.**
**Para la unidad 80h, esta función referencia la tabla de parámetros de disco duro apuntada por el vector de la interrupción 41h.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **06h** | |
| | **AL** | | **Factor de interleave.** |
| | **CH** | | **Número del cilindro.** |
| | **DH** | | **Número de cabeza** |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **07h** | **Número de unidad en DL es inválido.** |
| | | **01h** | **Solicitud de función inválida.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.14. Interfaz para la función 06h de Servicio BIOS a disco duro.

## Función 07h: Formatea unidad.

**Esta función formatea la unidad de disco duro completa, escribiendo campos de dirección de disco y sectores de datos, iniciando en el cilindro especificado. Esta función trabaja solo con controladores de disco duro de tipo XT.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **07h** | |
| | **AL** | | **Factor de interleave.** |
| | **CH** | | **Número del cilindro.** |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **07h** | **Número de unidad en DL es inválido.** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.15. Interfaz para la función 07h de Servicio BIOS a disco duro.

**Si ocurre algún error al invocar esta función, repita por lo menos dos veces intercalando llamadas a la función 00h (Inicialización) entre llamada y llamada.**

## Función 08h: Lee parámetros de unidad.

**Esta función retorna parámetros asociados con la unidad de disco duro (ya sea 80h u 81h) especificada en DL.**
**Para la unidad 80h, esta función referencia la tabla de parámetros de disco duro apuntada por el vector de la interrupción 41h. Para la unidad 81h, la función referencia al vector de la interrupción 46h. La sección dedicada a los datos de BIOS en memoria ROM de este mismo capítulo muestra el formato de la tabla de parámetros.**
**Si la llamada tiene éxito (se formatea correctamente la pista), la función retorna con el registro AL = 0.**
**El máximo número de cilindro utilizable es retornado en CH/CL, el máximo número de sector utilizable en CL, el máximo número de cabeza utilizable en DH, el número de unidades de disco duro instaladas en el sistema en DL, y la dirección de la tabla de parámetros de disco duro en ES:DI (formato Segmento:Offset).**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **08h** | |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **07h** | **Número de unidad en DL es inválido.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **AL** | **00h** | |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |
| | **CH** | | **Máximo número de cilindro utilizable (los 8 bits bajos)** |
| | **CL** | | **Número de Cilindro/Sector, donde:** |
| | | | **Bits 7 al 0 = 00h si AH = 07h.** |
| | | | **Bits 7 al 6 = Máximo número de cilindro utilizable (los 2 bits altos).** |
| | | | **Bits 5 al 0 = Máximo número de sector utilizable.** |
| | **DH** | | **Máximo número de cabeza utilizable.** |
| | **DL** | | **Número de unidades.** |
| | **ES:DI** | | **Dirección de la tabla de parámetros de disco duro.** |

Tabla E.16. Interfaz para la función 08h de Servicio BIOS a disco duro.

**Si el número de unidad especificado en DL no es válido, esta función retorna con AH y el Byte de Estado de disco duro (Dirección 40:74h) almacenando un valor de 07h para indicar la falla. Los registros AL, CX y DX son puestos en cero y el bit Carry Flag en 1.**

## Función 09h: Inicializa parámetros de unidad.

**Esta función inicializa el controlador asociado con la unidad de disco duro especificada en el registro DL (80h para la primera unidad física, 81h para la segunda).**
**Para la unidad 80h esta función referencia la tabla de parámetros de disco duro apuntada por el vector correspondiente a la interrupción 41h. Para la unidad 81h, la función utiliza el vector para la interrupción 46h. Vea la sección dedicada a los datos de BIOS en memoria ROM en este mismo apéndice para una descripción detallada del formato y contenido de la tabla de parámetros.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **09h** | |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **07h** | **La operación falló.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.17. Interfaz para la función 09h de Servicio BIOS a disco duro.

## Función 0Ah: Lectura larga de sectores.

**Esta función lee uno o más sectores de la unidad de disco duro especificada en el registro DL. También lee los de 4 a 7 bytes del código de corrección de error (ECC, Error Correction Code) asociado con ese sector, pero no corrige automáticamente errores de ECC (La función 02h si corrige automáticamente ese tipo de errores).**
**Las transferencias de sectores múltiples son terminadas en cuanto cualquier marque un error de lectura.**
**Los números de cilindro, sector y cabaza son especificados en los registros CH, CL y DH respectivamente.**
**Esta función no requiere ninguna llamada previa a la función de búsqueda (función 0Ch, la cual se describirá más adelante).**
**El uso de esta función está normalmente reservado a la realización de diagnósticos y no debiera ser usado para realizar lecturas normales desde el disco duro.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **0Ah** | |
| | **AL** | | **Número de sectores (usualmente uno).** |
| | **CH** | | **Número del cilindro (los 8 bits bajos)** |

| | | | |
|---|---|---|---|
| | CL | | Número de Cilindro/Sector, donde: |
| | | | Bits 7 al 6 = Número del cilindro (los 2 bits altos). |
| | | | Bits 5 al 0 = Número del sector. |
| | DH | | Número de la cabeza. |
| | DL | | Número de unidad física: |
| | | | 80h: Disco duro 1. |
| | | | 81h: Disco duro 2. |
| | ES:BX | | Dirección del buffer para realizar la transferancia. |
| Salida | AH | 00h | Sin errores. |
| | | XXh | Código del error |
| | | | Estado de disco fijo (Dirección 40h:74h). |
| | CF | 0 | Sin errores. |
| | | 1 | Hubo error. |

Tabla E.18. Interfaz para la función 0Ah de Servicio BIOS a disco duro.

## Función 0Bh: Escritura larga de sectores.

**Esta función escribe uno o más sectores en la unidad de disco duro especificada en el registro DL, tomando los datos del buffer apuntado por los registros ES:BX (formato Segmento:Offset). Esta función también escribe los de 4 a 7 bytes asociados con el Código de corrección de error (ECC, Error Correction Code) asociado con los sectores especificados.**

**Los números de cilindro, sector y cabeza son especificados en los registros Ch, CL y DH respectivamente.**

**Esta función no requiere la realización de una llamada a la función de búsqueda de cilindro (función 0Ch, la cual se explica más adelante).**

**Junto con la función 0Ah, esta función está por lo común restringida a operaciones de diagnóstico y no debería ser ussda para operaciones normales de escritura a disco duro.**

| | | | |
|---|---|---|---|
| Entrada | AH | 0Bh | |
| | AL | | Número de sectores a escribir (normalmente uno). |
| | CH | | Número del cilindro (los 8 bits bajos) |
| | CL | | Número de Cilindro/Sector, donde: |
| | | | Bits 7 al 6 = Número del cilindro (los 2 bits altos). |
| | | | Bits 5 al 0 = Número del sector. |
| | DH | | Número de la cabeza. |
| | DL | | Número de unidad física: |
| | | | 80h: Disco duro 1. |
| | | | 81h: Disco duro 2. |
| | ES:BX | | Dirección del buffer para realizar la transferencia. |
| Salida | AH | 00h | Sin errores. |
| | | XXh | Código del error |
| | | | Estado de disco fijo (Dirección 40h:74h). |

| | | | |
|---|---|---|---|
| | CF | 0 | Sin errores. |
| | | 1 | Hubo error. |

Tabla E.19. Interfaz para la función 0Bh de Servicio BIOS a disco duro.

## Función 0Ch: Busca cilindro.

**Esta función posesiona la cabeza de Lectura/Escritura sobre el cilindro especificado en los registros CH y CL. El número de disco físico debe ser especificado en el registro DL.**

**Las funciones:**

- **Lee sectores de disco (función 02h).**
- **Escribe sectores de disco (función 03h).**
- **Lectura larga de sectores (función 0Ah).**
- **Escritura larga de sectores (función 0Bh).**

**Realizan de manera implícita una búsqueda, cuyo código tienen interconstruido y no requieren una llamada previa a esta función.**

| | | | |
|---|---|---|---|
| Entrada | AH | 0Ch | |
| | CH | | Número del cilindro (los 8 bits bajos) |
| | CL | | Número de Cilindro/Sector, donde: |
| | | | Bits 7 al 6 = Número del cilindro (los 2 bits altos). |
| | | | Bits 5 al 0 = Número del sector. |
| | DH | | Número de la cabeza. |
| | DL | | Número de unidad física: |
| | | | 80h: Disco duro 1. |
| | | | 81h: Disco duro 2. |
| Salida | AH | 00h | Sin errores. |
| | | XXh | Código del error |
| | | | Estado de disco fijo (Dirección 40h:74h). |
| | CF | 0 | Sin errores. |
| | | 1 | Hubo error. |

Tabla E.20. Interfaz para la función 0Ch de Servicio BIOS a disco duro.

## Función 0Dh: Inicialización alterna de disco duro.

**Esta función es idéntica a la función 00h (Inicialización), excepto porque el sistema de disco flexible no es inicializado.**

**El programa que llama a esta función debe especificar la unidad de disco a inicializar en el registro DL.**

**Esta función reinicializa la tarjeta controladora de disco duro y coloca la unidad en un estado conocido inicializando los parámetros de disco duro y recalibrando las posiciones de las cabezas de Lectura/Escritura a la pista 0.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **0Dh** | |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.21. Interfaz para la función 0Dh de Servicio BIOS a disco duro.

## Función 0Eh: Diagnósticos 1: Lee buffer de prueba.

**Esta función lee un buffer de prueba desde la tarjeta controladora de disco duro y deposita el resultado en el buffer especificado por los registros ES:BX (format Segmento:Offset). Los datos en realidad no son leídos desde la unidad de disco físico.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **0Eh** | |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| | **ES:BX** | | **Apuntador al buffer de diagnóstico.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.22. Interfaz para la función 0Eh de Servicio BIOS a disco duro.

## Función 0Fh: Diagnósticos 2: Escribe buffer de prueba.

**Esta función escribe un buffer de prueba a la tarjeta controladora de disco duro. La dirección del buffer está dada por los registros ES:BX (formato Segmento:Offset). Los datos no se escriben a la unidad de disco físico.**

**Esta función debería ser llamada para inicializar los contenidos del buffer de sector antes de formatear una unidad de disco tipo XT utilizando la función 05h.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **0Fh** | |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| | **ES:BX** | | **Apuntador al buffer de diagnóstico.** |
| **Salida** | **AH** | **00h** | **Sin errores** |
| | | **XXh** | **Código del error** |
| | | | **Estado da disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.23. Interfaz para la función 0Fh de Servicio BIOS a disco duro.

## Función 10h: Prueba si la unidad está lista.

**Esta función determina si la unidad de disco duro especificada por el registro DL está lista y puede procesar un comando. Si la función falla ésto implica que la unidad se ancuentra aún procesando otros comandos. Si esta función se llama en unidades de disco flexible se puede averiguar si la unidad contiene un disco dentro o no.**

**La tabla especificando la interfaz de llamada a esta función es la E.24.**

## Función 11h: Recalibra la unidad.

**Esta función reposiciona la cabeza de Lectura/Escritura 0 sobre el cilindro 0 de la unidad da disco duro especificada en el registro DL.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **10h** | |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| | **ES:BX** | | **Apuntador al buffer de diagnóstico.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 4Jh:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.24. Interfaz para la función 10h de Servicio BIOS a disco duro.

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **11h** | |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.25. Interfaz para la función 11h de Servicio BIOS a disco duro.

## Función 12h: Diagnóstico de memoria RAM del controlador.

**La llamada a esta función indica a la tarjeta controladora de disco duro que realice un diagnóstico interconstruido sobre su buffer interno de sector, indicando si le prueba fue aprobada mediante el valor almacenado en el registro AH al retornar de la función.**

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **12h** | |
| | **AL** | | **Número de sectores.** |
| | **CH** | | **Cilindro.** |
| | **CL** | | **Sector.** |
| | **DH** | | **Cabeza.** |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.26. Interfaz para la función 12h de Servicio BIOS a disco duro.

## Función 13h: Diagnóstico de la unidad del controlador.

**Esta función indica a la tarjeta controlsdora de disco duro la realización de diagnósticos interconstruidos a la unidad de diaco duro unida al sistema.**

**El valor guardado en el registro AH al retornar de la llamada indica si el diagnóstico fue aprobado.**

**Si un parámetro no válido es pasado en uno de los registros de entrada, el bit de Carry Flag se prende y el control se retorna al programa que hizo la llamada, dejando los valores en los registros de entrada intactos.**

| Entrada | AH | 13h | |
|---|---|---|---|
| | AL | | Número de sectores. |
| | CH | | Cilindro. |
| | CL | | Sector. |
| | DH | | Cabeza. |
| | DL | | Número de unidad física: |
| | | | 80h: Disco duro 1. |
| | | | 81h: Disco duro 2. |
| Salida | AH | 00h | Sin errores. |
| | | XXh | Código del error |
| | | | Estado de disco fijo (Dirección 40h:74h). |
| | CF | 0 | Sin errores. |
| | | 1 | Hubo error. |

Tabla E.27. Interfaz para la función 13h de Servicio BIOS a disco duro.

## Función 14h: Diagnóstico interno del controlador

**Esta función es invocada para indicar a la tarjeta controladora de disco duro la realización de un diagnóstico interconstruido, el cual se aplica a si misma.**
**El resultado de dicho diagnóstico es indicado al retornar de la función en el registro AH.**
**Si un parámetro ilegal es pasado en uno de los registros de entrada el bit Carry Flag se prende y el control es pasado al programa que invocó la función, preservando los registros de entrada para permitir examinar la información que contienen.**
**La Tabla que define la interfaz a esta función es la E.28.**

## Función 15h: Lee tipo de disco duro.

**Esta función retorna un código de estado en el registro AH, el cual indica el tipo de unidad de disco flexible o duro para el identificador almacenado en el registro DL.**
**A su vez, esta función retorna el número de bloques de 512 bytes (sectores) que existen en la unidad de disco duro si la especificación en DL es válida (Los valores aceptables son 80h para la primera unidad física y 81h para la segunda).**
**Si la llamada a esta función es exitosa, la palabra alta del número que indica la cantidad de bloques de 512 bytes (sectores) de la unidad de disco duro es retornada en el registro CX, y la palabra baja es retornada en el registro DX. Se almacena un cero en el byte de estado de disco duro (Dirección 40:74h) y en el registro AH, se limpia el bit de Carry Flag, y se retorna el control al programa que invocó la función.**
**Para el número de unidad física 80h, esta función referencia la tabla de parámetros de disco duro apuntada por el vector correspondiente a la interrupción 41h.**

Para el número de unidad física 81h, esta función referencia la tabla de parámetros de disco duro apuntada por el vector correspondiente a la interrupción 46h.
A diferencia de muchas otras funciones de servicio a disco duro, en esta función el valor almacenado en el registro AH no es igual al valor almacenado en el byte de Estado de disco duro (Dirección 40:74h). En vez de ello, el registro AH reporta si la entrada de número de unidad almacenada en el registro DL es válida (AH será igual a cero si no hay unidad de disco duro instalada para el número de unidad proporcionado en el registro DL), o el estado de la operación realizada.
Si el número de unidad especificado en el registro DL no es válido, esta función almacena un cero en los registros AH, AL, CX, DX y en el byte de Estado de disco duro (Dirección 40:74h), limpia el bit de Carry Flag y retorna el control al programa que emitió la llamada. La Tabla que define su interfaz es la E.29.

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **14h** | |
| | **AL** | | **Número de sectores.** |
| | **CH** | | **Cilindro.** |
| | **CL** | | **Sector.** |
| | **DH** | | **Cabeza.** |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **Sin errores.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.28. Interfaz para la función 14h de Servicio BIOS a disco duro.

| | | | |
|---|---|---|---|
| **Entrada** | **AH** | **15h** | |
| | **DL** | | **Número de unidad física:** |
| | | | **80h: Disco duro 1.** |
| | | | **81h: Disco duro 2.** |
| **Salida** | **AH** | **00h** | **No hay unidad de disco instalada.** |
| | | **03h** | **El disco duro fue accesado.** |
| | | **XXh** | **Código del error** |
| | | | **Estado de disco fijo (Dirección 40h:74h).** |
| | **CX** | | **Número de bloques de 512 bytes (sectores) en la unidad de disco duro (La palabra alta). Válido solo si AH=3.** |
| | **DX** | | **Número de bloques de 512 bytes (sectores) en la unidad de disco duro (La palabra baja). Válido solo si AH=3.** |
| | **CF** | **0** | **Sin errores.** |
| | | **1** | **Hubo error.** |

Tabla E.29. Interfaz para la función 15h de Servicio BIOS a disco duro.

# Apéndice F. Formato de la interfaz a un Controlador de dispositivos.

## Introducción.

**Dos utiliza una estructura conocida como Encabezado de Solicitud (Request Header) para pasar instrucciones a un Controlador de dispositivos y recibir resultados de éste. Sin embargo, dicha estructura no tiene un formato fijo. Dicho formato varia dependiendo del comando que se esté enviando al Controlador. Ésta técnica provee un medio de comunicación flexible y eficiente. Sin embargo, Se vuelve un tanto complejo el dominio de todos los formatos existentes. Es por ésto que a pesar de que la información necesaria para conocer estos mecanismos ya fue proporcionada a lo largo de este trabajo se decidió cubrirla nuevamente dentro de este apéndice, que se buscó fuera lo más conciso posible.**

**Existe un conjunto de caracteristicas qua todos los comandos poseen. Es por ello que los diseñadores de esta interfaz decidieron separar a un Encabezado de solicitud en dos partes: Una parte fija y una parte variante (Dependiendo del comando que se ejecute). Obviamente, habrá comandos a los que por su semejanza se les haya definido una parte variante idéntica.**
**Durante el resto de este apéndice se mostrarán tablas que indican el formato de la estructura Encabezado de Solicitud para la parte fija y para los diferentes comandos. En cada renglón de cada tabla se indica el nombre del campo, su longitud en bytes y su tipo. Es decir, si dicho campo es pasado como parámetro al controlador o si el controlador se encarga de asignarle un valor para pasarlo como resultado a quien lo invocó.**

### Parte Fija del Encabezado de solicitud.

| Descripción | Tamaño[1] | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Longitud de este Encabezado de Solicitud | 1 | | |
| Número de unidad[2] | 1 | | |
| Código del comando a ejecutar | 1 | | |
| Código del resultado de la operación | 2 | | |
| Reservado | 8 | | |

Tabla F.1. Parte no variante del Encabezado de Solicitud a Controlador de dispositivos.

[1] El tamaño del campo se proporciona en Bytes.
[2] Solo usado en dispositivos de bloque, los cuales pueden manejar más de un dispositivo al mismo tiempo. Por tanto, se hace necesario indicar el dispositivo al cual se dirige la instrucción. Un ejemplo de ésto es el controlador que se encarga de las unidad A:, B:, C: y D: de una microcomputadora.

## Comando 0 - *Initialization* (Inicialización).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Número de unidades que el controlador manejará[3] | 1 | | |
| Offset de la dirección en que DOS deberá cortar el código después de inicializar | 2 | | |
| Segmento de la dirección en que DOS deberá cortar el código después de inicializar | 2 | | |
| Offset de la dirección del apuntador al vector de estructuras de tipo BPB[4] | 2 | | |
| Segmento de la dirección del apuntador al vector de estructuras de tipo BPB[5] | 2 | | |
| Primera unidad disponible[6] | 1 | | |

Tabla F.2. Parte variante del comando 0 de un Controlador de dispositivos.

## Comando 1 - *Media Check* (Checa si el medio ha cambiado).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Descriptor de medio, leído del BPB de la unidad | 1 | | |
| Estado del medio, retornado por el controlador | 1 | | |
| Offset de la dirección en que se encuentra el identificador de volúmen | 2 | | |
| Segmento de la dirección en que se encuentra el identificador de volúman | 2 | | |

Tabla F.3. Parte variante del comando 1 de un Controlador de dispositivos.

## Comando 2 - *Get BIOS parameter Block* (Obten el bloque de parametros descriptores de dispositivo en BIOS).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Descriptor de medio, leído del BPB de la unidad | 1 | | |
| Offset de la dirección de transferencia de datos | 2 | | |
| Segmento de la dirección del área de transferencia de datos | 2 | | |
| Offset de la dirección del BPB | 2 | | |
| Segmento de la dirección al BPB | 2 | | |

Tabla F.4. Parte variante del comando 2 de un Controlador de dispositivos.

[3] Solo para Controladores de dispositivos tipo Bloque.
[4] Solo para Controladores de dispositivos tipo Bloque.
[5] Solo para Controladores de dispositivos tipo Bloque.
[6] Solo para Controladores de dispositivos tipo Bloque.

### Comando 3 - *IOCTL Input* (Lee información de control desde el dispositivo).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Descriptor de medio, leído del BPB de la unidad | 1 | | |
| Offset de la dirección del área de transferencia de datos | 2 | | |
| Segmento de la dirección del área de transferencia de datos | 2 | | |
| Tamaño de la transferencia[7] | 2 | | |
| Número del sector inicial[8] | 2 | | |

Tabla F.5. Parte variante del comando 3 de un Controlador de dispositivos.

### Comando 4 - *Input* (Lee datos del dispositivo).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Descriptor de medio, leído del BPB de la unidad | 1 | | |
| Offset de la dirección del área de transferencia de datos | 2 | | |
| Segmento de la dirección del área de transferencia de datos | 2 | | |
| Tamaño de la transferencia | 2 | | |
| Número del sector inicial | 2 | | |
| Offset de la dirección en que se encuentra el identificador de volúmen | 2 | | |
| Segmento de la dirección en que se encuentra el identificador de volúmen | 2 | | |
| Número del sector inicial, en formato 32 bits. | 4 | | |

Tabla F.6. Parte variante del comando 4 de un Controlador de dispositivos.

### Comando 5 - *Nondestructive Input*[9] (Lee datos sin borrar buffer).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Caracter leído del dispositivo | 1 | | |

Tabla F.7. Parte variante del comando 5 de un Controlador de dispositivos.

[7] En sectores si el dispositivo es de bloque y en caracteres si es de caracter.
[8] Solo para dispositivos de Bloque.
[9] Solo para dispositivos de Caracter.

## Comando 6 - *Input Status* (Averigua si se pueden leer datos).

**Este comando no requiere una estructura adicional. Solo retorna su Salida a través del byte de *Resultado de la operación* de la estructura que representa a la parte fija del Encabezado de Solicitud.**

## Comando 7 - *Input Flush*[10] (Limpia el buffer de entrada).

**Al igual que el anterior, este comando no requiere una estructura adicional para su operación. Indica el Resultado de su acción a través del byte de *Resultado de la operación* de la estructura que representa a la parte fija del Encabezado de Solicitud.**

## Comando 8 - *Output* (Escribe datos a dispositivo).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Descriptor de medio, leído del BPB de la unidad | 1 | | |
| Offset de la dirección del área de transferencia de datos | 2 | | |
| Segmento de la dirección del área de transferencia de datos | 2 | | |
| Tamaño de la transferencia[11] | 2 | | |
| Número del sector inicial[12] | 2 | | |
| Offset de la dirección en que se encuentra el identificador de volúmen | 2 | | |
| Segmento de la dirección en que se encuentra el identificador de volúmen | 2 | | |
| Número del sector inicial, en formato 32 bits[13] | 4 | | |

Tabla F.8. Parte variante del comando 8 de un Controlador de dispositivos.

## Comando 9 - *Output with Verify* (Escribe y checa escritura).

**El formato de la estructura variable para este comando es idéntico a la mostrada para el comando anterior. La única diferencia es que, si es posible, este comando**

[10] Solo para dispositivos de Caracter.
[11] Para dispositivos de Caracter se mide en bytes, para dispositivos de Bloque se mide en Sectores.
[12] Solo para dispositivos de Bloque.
[13] Cuando el Campo *Número de Sector inicial* es cero el disco excede 32 MegaBytes y este campo contiene el equivalente, en 4 bytes.

**debe leer lo escrito y compararlo con el original para verificar su correcta escritura.**

### Comando 10 - *Output Status*[14] (Averigua si se pueden escribir datos).

**Este comando no requiere una estructura adicional. Solo entrega su resultado a través del byte de *Resultado de la operación* de la estructura que representa a la parte fija del Encabezado de Solicitud.**

### Comando 11 - *Output Flush*[15] (Descarta información en buffer de escritura).

**Este comando no requiere una estructura adicional. Solo entrega su resultado a través del byte de *Resultado de la operación* de la estructura que representa a la parte fija del Encabezado de Solicitud.**

### Comando 12 - *IOCTL Output* (Envía información de control al controlador).

| | | Tipo | |
|---|---|---|---|
| **Descripción** | **Tamaño** | **Entrada** | **Salida** |
| Descriptor de medio, leído del BPB de la unidad | 1 | | |
| Offset de la dirección del área de transferencia de datos | 2 | | |
| Segmento de la dirección del área de transferencia de datos | 2 | | |
| Tamaño de la transferencia[16] | 2 | | |
| Número del sector inicial[17] | 2 | | |
| Offset de la dirección en que se encuentra el identificador de volúmen | 2 | | |
| Segmento de la dirección en que se encuentra el identificador de volúmen | 2 | | |
| Número del sector inicial, en formato 32 bits[18] | 4 | | |

Tabla F.9. Parte variante del comando 12 de un Controlador de dispositivos.

[14] Solo se envía a Controladores de dispositivos de Caracter.
[15] Este comando se envía solo a Controladores de dispositivos de tipo Caracter.
[16] En sectores para dispositivos de bloque, en bytes para dispositivos de tipo Caracter.
[17] Solo para dispositivos de Bloque.
[18] Cuando el Campo *Número de Sector inicial* es cero el disco excede 32 MegaBytes y este campo contiene el equivalente, en 4 bytes.

### Comando 13 - *Device Open* (Abre dispositivo).

**Este comando no requiere una estructura adicional. Solo entrega su resultado a través del byte de *Resultado de la operación* de la estructura que representa a la parte fija del Encabezado de Solicitud.**

### Comando 14 - *Device Close* (Cierra dispositivo).

**Este comando no requiere una estructura adicional. Solo entrega su resultado a través del byte de *Resultado de la operación* de la estructura que representa a la parte fija del Encabezado de Solicitud.**

### Comando 15 - *Removable Media*[19] (Verifica si el medio es removible).

**Este comando no requiere una estructura adicional. Solo entrega su resultado a través del byte de *Resultado de la operación* de la estructura que representa a la parte fija del Encabezado de Solicitud.**

### Comando 16 - *Output Until Busy*[20] (Escribe hasta que el buffer de escritura se llene).

| | | Tipo | |
|---|---|---|---|
| **Descripción** | **Tamaño** | **Entrada** | **Salida** |
| **Descriptor de medio, leído del BPB de la unidad** | 1 | | |
| **Offset de la dirección del área de transferencia de datos** | 2 | | |
| **Segmento de la dirección del área de transferencia de datos** | 2 | | |
| **Número deBytes escritos por el Controlador** | 2 | | |

Tabla F.10. Parte variante del comando 16 de un Controlador de dispositivos.

### Comando 17 - Reservado.

### Comando 18 - Reservado.

[19] Solo para dispositivos de Bloque.
[20] Solo para dispositivos tipo Caracter.

### Comando 19 - *Generic IOCTL*[21] (Control de Entrada/Salida Estándar).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Función mayor | 1 | | |
| Función menor | 1 | | |
| Contenido del registro SI | 2 | | |
| Contenido del registro DI | 2 | | |
| Offset de la dirección de la Solicitud | 2 | | |
| Segmento de la dirección de la Solicitud | 2 | | |

Tabla F.11. Parte variante del comando 19 de un Controlador de dispositivos.

### Comando 20 - Reservado
### Comando 21 - Reservado
### Comando 22 - Reservado
### Comando 23 - *Get Logical Device*[22] (Obten la letra que representa a un dispositivo de bloque).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| • En entrada: Código de la unidad<br>• En salida: Último dispositivo | 1 | | |
| Código del comando | 1 | | |
| Resultado | 2 | | |
| Reservado | 4 | | |

Tabla F.12. Parte variante del comando 23 de un Controlador de dispositivos.

### Comando 24 - *Set Logical Device*[23] (Designa un dispositivo con una letra especifica).

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| • En entrada: Código de la unidad<br>• En salida: Último dispositivo | 1 | | |
| Código del comando | 1 | | |
| Resultado | 2 | | |
| Reservado | 4 | | |

Tabla F.13. Parte variante del comando 24 de un Controlador de dispositivos.

[21] Solo para dispositivos de Bloque.
[22] Solo para dispositivos de Bloque.
[23] Solo para dispositivos de Bloque.

**Comando 25 - *IOCTL Query* (Interroga acerca del control de E/S estándar).**

| Descripción | Tamaño | Tipo | |
|---|---|---|---|
| | | Entrada | Salida |
| Función mayor | 1 | | |
| Función menor | 1 | | |
| Contenido del registro SI | 2 | | |
| Contenido del registro DI | 2 | | |
| Offset de la dirección de la Solicitud | 2 | | |
| Segmento de la dirección de la Solicitud | 2 | | |

Tabla F.14. Parte variante del comando 25 de un Controlador de dispositivos.

# Glosario

**80x86**
Denominación de Familia de microprocesadores de la compañía Intel que comprende al 80286, 80386 y 80486.

**API**
Siglas de *Application Programming Interface* (Interfaz para programación de aplicaciones). Es el término con que se conoce al conjunto de programas y estándares diseñados e implementados para facilitar a un desarrollador de programas la creación de aplicaciones para un ambiente específico. Se puede aplicar al conjunto de rutinas BIOS proporcionadas por una microcomputadora para accesar sus recursos.

**Apple**
Compañía Norteamericana fundada a mediados de la década de los setentas que introdujo en 1984 su computadora personal Macintosh, la cual estaría destinada a revolucionar la industria del cómputo gracias a sus innovadores conceptos de interfaz amigable a usuario y características de gran valía técnica.

**Arbitraje**
Proceso a través del cual los dispositivos de un equipo de cómputo compiten por la posesión de un canal de comunicaciones, en base a prioridades.

**Arbitraje, Nivel de**
Un nivel de Arbitraje es el nivel de Prioridad asignado a dispositivos que estén sujetos a un proceso de Arbitraje.

**ASCII, Código**
Siglas de *American Standard Code for Information Exchange* (Código Americano estándar para el intercambio de información), un código formado por elementos de caracteres, control y gráficos de 7 bits cada uno. Existen versiones de código ASCII Extendido que utilizan 8 bits.

**BIOS (Basic Input/Output System, Sistema básico de Entrada/Salida)**
Software de sistema que se encarga de realizar una interfaz entre el Sistema Operativo y el Hardware de una computadora.

**BIOS, Servicio**
Una rutina de software que sirve a un dispositivo periférico dado, y provee una interfaz entre el Sistema operativo y dicho dispositivo. Estos servicios (En DOS) son funciones simples unitarias.

**Bit**
Es la más pequeña unidad de información que un equipo de cómputo digital puede manejar. Puede tomar solo uno de dos valores: 0 o 1.

**Bits por segundo**
Una unidad de medida que representa la cantidad de bits transmitidos por un dispositivo determinado en un segundo.

**Borland**
Compañía Norteamericana de Software fundada por Philipe Kahn. Se dio a conocer por su Compilador Turbo Pascal versión 1.0. Varios productos de esta empresa se utilizan para desarrollar la aplicación presentada en este trabajo, como Turbo C++ versión 1.0, Tlink y Tlib.

**BPB**
Siglas que significan BIOS Parameter Block (Bloque de parámetros BIOS). El BPB es una estructura de datos que forma parte del Registro de Arranque (Boot sector). Su objetivo es permitir a DOS comprender como es el disco en el cual se encuentra dicho BPB. Contiene información acerca de los parámetros físicos del disco (Números de cabezas, etcétera), así como de la estructura del Sistema de archivos DOS que tiene instalado.

**Buffer**
Un área de almacenamiento en memoria (Principal o Secundaria) que está temporal o permanentemente reservada para realizar operaciones de Entrada/Salida.

**Bus**
En una computadora, un conjunto de una o más líneas conductoras que transportan niveles de voltaje que se consideran como señales lógicas.

**Bus de direcciones**
En una computadora, un conjunto de una o más líneas conductoras que transportan desde el microprocesador a las demás partes del sistema niveles de voltaje, y que se interpretan como códigos de dirección.

**Byte**
Ocho bits contiguos. Es la unidad de información que una microcomputadora usa para representar un caracter.

**C, Lenguaje**
Lenguaje de alto nivel para el desarrollo de aplicaciones en computadoras que fue creado por Brian Kernighan y Dennis Ritchie inspirandose en el lenguaje BCPL, del cual toma muchas de sus caracteristicas. En 1973 fue usado para reescribir parte del código del Sistema Operativo UNIX.

**Cabeza de Lectura/Escritura (Head, I_O Head)**
Es el dispositivo magnetizable que provee el acceso a la superficie magnética de un disco duro o flexible.

**Canal (Channel)**
En una microcomputadora, una ruta de hardware construida especificamente para transportar señales.

**CGA**
Siglas de *Color Graphics Adapter*, la cual es una técnica de manejo de video en una microcomputadora. Sus caracteristicas permiten presentar 4 colores simultáneamente en pantalla.

**Cilindro (Cylinder)**
Un Cilindro está formado por el conjunto de Pistas de un disco duro que pueden ser accesadas a la vez por todas las cabezas de Entrada/Salida, sin mover el peine de Lectura/Escritura.

**Cluster**

**En este texto se traduce como *Grupo de Sectores*. Es una denominación usada en el Sistema Operativo DOS para dividir a un disco duro o flexible. Es la mínima cantidad de espacio que se puede asignar a un archivo. Si el archivo es menor que un *Cluster*, el espacio sobrante se desperdicia. Sin embargo, esta técnica permite destinar menos espacio a las áreas de administración de disco y hacer más veloz su acceso.**

**CMOS**

**Siglas de *Complementary Metal Oxide Semiconductor* (Semiconductor complementario de Metal y Óxido). Es una tecnología usada entre otras cosas para desarrollar memorias digitales de bajo consumo. En una microcomputadora, memoria de este tipo es utilizada para almacenar información de configuración básica del equipo y está alimentada por una pequeña batería, para evitar que pierda su contenido cuando el equipo se apague.**

**Config.Sys**

Archivo de texto convencional que es usado en DOS para configurar una microcomputadora. Cuando el sistema se arranca dicho archivo es leído e interpretado por el código de inicialización. La información que contiene permite a un equipo reconocer nuevos dispositivos y manejar los ya conocidos de una manera más eficiente.

**Corrimiento (Offset)**

En el Direccionamiento de memoria en una microcomputadora, es un método que consiste en definir una dirección como un corrimiento a partir de un segmento.

**CP/M**

Siglas de *Control Program for Microcomputers* (Programa de Control para Microcomputadoras), el cual fue uno de los Sistemas Operativos pioneros en la Microcomputación. Inició como un Sistema para computadoras de 8 bits. Su versión de 16 bits fue inicialmente considerada como el Sistema Operativo básico para las primeras microcomputadoras tipo PC (de IBM).

**CRC**

**Siglas de *Cyclic Redundancy Check* (Chequeo de Redundancia cíclica)**
**Un método de chequeo de redundancia donde la llave de prueba es generada por un algoritmo iterativo. Método común de chequeo de errores en información.**

**Cylinder, vease Cilindro.**

**Device Close**

**Uno de los comandos que DOS puede enviar a un Controlador de dispositivos. Es el comando número 14 y está definido para las versiones de DOS 3.0 o posteriores. Este comando es usado en coordinación con el comando *Device Open* para controlar el número de veces que un dispositivo es abierto o cerrado.**

**Device Header**
El Device Header (Encabezado de dispositivo) es la primera parte de un controlador de dispositivos. Define características básicas acerca del comportamiento de éste. Contiene campos que apuntan a las rutinas Estrategia e Interrupción del controlador, el nombre del controlador, los bytes de atributo del controlador, y un apuntador al siguiente controlador de la cadena de controladores del sistema.

**Device Open**
Es el comando número 13 de los que DOS envía a un Controlador de dispositivos. Disponible a partir de la versión de DOS 3.0. Este comando se usa en conjunto con el comando *Device Close* para controlar el número de veces que un dispositivo es abierto o cerrado. También es común usarlo para reinicializar el dispositivo o para restringir usos múltiples de un dispositivo.

**Disco duro (Fixed disk, Hard disk)**
Un dispositivo de almacenamiento magnético consistente de un mecanismo giratorio con uno o más discos metálicos permanentemente montados.

**DMA**
Siglas de Direct Access Memory (Memoria de acceso directo), es una técnica mediante la que los dispositivos de Entrada/Salida de una microcomputadora transfieren datos directamente a y desde la memoria principal del sistema sin intervención del microprocesador. Ésto decrementa significativamente el tiempo que el microprocesador permanece ocupado por tareas de Entrada/Salida.

**DMA, Controlador de**
Un controlador de DMA es un dispositivo que emite códigos de direcciones y control al dispositivo que en ese momento haya ganado el *bus* mediante un proceso de *Arbitraje*. Este controlador no interviene en el proceso de arbitraje.

**DMA, Dispositivo**
Un dispositivo DMA posee la capacidad de realizar procesos de Entrada/Salida sin la intervención del microprocesador central. Un dispositivo de este tipo participa en el proceso de *Arbitraje* para usar el *Canal*. Cuando lo consigue, recibe códigos de dirección y control desde el *Controlador de DMA* para poder escribir y leer información.

**DOS (Disk operating system)**
Sistema operativo para microcomputadoras del tipo PC o compatibles.

**ECC**
Siglas de Error correction Code (Código de corrección de error). Es una técnica usada para la detección y corrección de errores a nivel bit, la cual utiliza un algoritmo que prueba la precisión de los datos transmitidos.

**Ensamblador, Lenguaje**
Lenguaje de programación de bajo nivel que es altamente dependiente del hardware del sistema al que pertenezca. Por sus características de velocidad y control de hardware es adecuado para desarrollar aplicaciones con mucha interrelación al sistema.

**FAT**
Siglas de *File allocation Table* (Tabla de Registro de Archivos). Es la técnica usada por el Sistema operativo DOS para administrar la asignación de espacio de disco a archivos.

**Firmware**
En contraposición a Software y Hardware, Nombre con el que se designa el software escrito para ser permanentemente grabado en una computadora, y que está intimamente ligado a la forma en que trabaja el hardware.

**Fixed disk, vease disco duro**

**Generic IOCTL**
Comando 19 de la interfaz DOS a Controlador de dispositivos. Es solo válido para dispositivos de bloque cuando se usa la versión de DOS 3.2 o posterior. Sin embargo, DOS 3.3 permite este comando también para dispositivos de caracter. Su propósito es proveer servicios estándares de control de Entrada/Salida a dispositivos.

**Get BIOS parameter Block**
Comando 2 del conjunto de comandos emitidos por DOS a un Controlador de dispositivos. Se aplica solo a dispositivos de bloque, como discos duros o flexibles, unidades lectoras de CD-ROM, etcétera. Su objetivo es el de provaer al Sistema operativo con el *Bloque de parámetros BIOS* (BPB) del dispositivo para permitirle tomar decisiones a partir de la información proporcionada.

**Get Logical Device**
Comando 23 de la interfaz DOS a Controlador de dispositivos. Este comando está disponible solo para dispositivos de bloque cuando se ejecuta la versión de DOS 3.2 o posterior. A partir de la versión 3.2 de DOS se permite utilizar más de una letra de unidad lógica para accesar un mismo dispositivo fisico. Este comando permite averiguar cuales letras se han usado para accesar el dispositivo controlado.

**GigaByte**
Unidad de medida que equivale a $2^{30}$ (2 elevado a la potencia 30) bytes. En otras palabras, 1,073'741,824 bytes-Más de Mil millones de bytes.

**Hard disk, vease disco duro**

**Hardware**
Conjunto de componentes fisicos de un equipo de cómputo.

**Head, I/O Head, vease Cabeza**

**Hexadecimal, numeración**
Notación utilizada para representar números. Por sus caracteristicas permite ver cifras mayores con menos digitos que la numeración decimal. Además, es muy conveniente para representar números que son potencias de 2. Por ésto, es ampliamente usado en Computación. Posee 16 digitos, los cuales son los 10 digitos decimales (0 al 9) y las primeras 6 letras del alfabeto (A a F).

**IBM**
Siglas de International Business Machines, Empresa dedicada a la Computación fundada a principios de siglo en Estados Unidos.

**Inicialization**

Dentro del conjunto de comandos que DOS envía a un Controlador de Dispositivos, el comando principal es el de Inicialización. Siempre es invocado tan pronto como el controlador es cargado en memoria. Su objetivo es permitir al Controlador realizar acciones básicas de preparación para los demás comandos o, en su caso, cancelar la instalación.

**Input**

Es el comando cuarto dentro de los que DOS envía a un Controlador de Dispositivos. Su objetivo es proporcionar al sistema operativo datos leídos desde el dispositivo controlado. Se encarga de realizar las operaciones de bajo nivel necesarias para ésto.

**Input Flush**

Dentro de la interfaz de DOS a los Controladores de Dispositivos, es el comando siete. Su función es limpiar el buffer de entrada. Comúnmente se usa para borrar cualquier información extraña antes de realizar una entrada de datos de importancia. De esta forma se evita cualquier error por datos indeseados.

**Input Status**

Es el comando 6 de los que un Controlador de dispositivos puede ejecutar a petición del Sistema operativo. Su función es checar que el dispositivo controlado esté listo para ser leído. Típicamente es enviado por DOS antes de realizar enviar un comando *Input*.

**Intel**

Empresa Norteamericana creadora y proveedora de una gran parte de los microcontroladores usados por las microcomputadoras PC y compatibles. Sus productos que se han usado para el mercado de PC's son el 8088/8086, 80286, 80386, 80486 y al Pentium (En sus diversas modalidades).

**InterLink**

Utilería incluida a partir de la versión 6.0 del Sistema operativo DOS que permite ver un recurso (Unidad de disco duro o flexible o puerto de impresión) de una microcomputadora remota (comunicada por puerto serial o parsielo) mediante un designador de unidad lógica en la microcomputadora local. Su interés para este trabajo es que la técnica que utiliza para ver los recursos remotos es la misma que el Controlador desarrollado para este trabajo usa.

**Interrupción**

Se conoce como interrupción a la Suspensión momentánea de la ejecución de un programa por parte del Microprocesador para atender una demanda de servicio proveniente de un dispositivo periférico. Después de que la demanda de servicio ha sido satisfecha la tarea suspendida es reanudada en el punto en que fuera interrumpida.

**IntrSpy**

Sistema de depuración de código que está basado en eventos y en el uso de Scripts. Por ésto último, es muy adecuado como auxiliar en el desarrollo y depuración de Controladores de dispositivos, en los cuales es muy difícil usar depuradores tradicionales.

**IOCTL Input**
Tercer Comando entre los definidos para el manejo de un Controlador de dispositivos. IOCTL Significa *Input/Output Control* (Control de Entrada/Salida). Este comando es utilizado para que el Controlador retorne información importante acerca del dispositivo controlado. Por ejemplo, si el dispositivo es una impresora conectada a un puerto serial, se podría retornar información acerca de la velocidad del canal.

**IOCTL Output**
Es el comando número 13 de los que DOS define para el manejo de un Controlador de dispositivos. IOCTL Significa Input/Output Control (Control de Entrada/Salida). Su objetivo es enviar información de manejo al controlador. El controlador no escribe nada al dispositivo, solo utiliza esa información para modificar su conducta.

**IOCTL Query**
Comando 26 de la Interfaz DOS a Controlador de dispositivos. Este comando está disponible para dispositivos de cualquier tipo corriendo bajo la versión 5.0 o posterior de DOS. Este comando es utilizado por un programa para interrogar a un controlador acerca de si soporta una determinada función genérica de IOCTL.

**Kilobyte**
Unidad de medida equivalente a $2^{10}$ bytes. Es decir, 1,024 bytes.

**Macintosh**
Computadora desarrollada por la Empresa *Apple Computer*, la cual sentó precedentes en cuanto a ambientes de cómputo amigables. Su Filosofía es actualmente imitada por sistemas competidores.

**MASM**
Ensamblador de código de la empresa Microsoft, que inicialmente era embarcado con su Sistema Operativo DOS.

**Media Check**
Comando 1 en la interfaz definida por DOS para la comunicación con Controladores de dispositivos. Se aplica solo a dispositivos de bloque, como unidades de disco duro o flexible, unidades lectoras de disco compacto, etcétera. Su función es la de avisar al Sistema operativo si el dispositivo ha cambiado desde la última llamada. Su utilidad se manifiesta principalmente con las unidades de disco flexible, pues indica a DOS que se removió el último disco en usarse y se insertó otro. Típicamente, DOS enviaría en ese caso el comando 2, Get BPB, para conocer las características físicas y lógicas del nuevo dispositivo.

**Megabyte**
Unidad de medida equivalente a $2^{20}$ bytes. Es decir, 1'048,576 bytes.

**Memoria principal**
En una microcomputadora, es la memoria RAM que se localiza entre 0 y 1 MegaByte. Su sinónimo es Memoria convencional.

**IOCTL Input**
Tercer Comando entre los definidos para el manejo de un Controlador de dispositivos. IOCTL significa *Input/Output Control* (Control de Entrada/Salida). Este comando es utilizado para que el Controlador retorne información importante acerca del dispositivo controlado. Por ejemplo, si el dispositivo es una impresora conectada a un puerto serial, se podría retornar información acerca de la velocidad del canal.

**IOCTL Output**
Es el comando número 13 de los que DOS define para el manejo de un Controlador de dispositivos. IOCTL significa Input/Output Control (Control de Entrada/Salida). Su objetivo es enviar información de manejo al controlador. El controlador no escribe nada al dispositivo, solo utiliza esa información para modificar su conducta.

**IOCTL Query**
Comando 26 de la interfaz DOS a Controlador de dispositivos. Este comando está disponible para dispositivos de cualquier tipo corriendo bajo la versión 5.0 o posterior de DOS. Este comando es utilizado por un programa para interrogar a un controlador acerca de si soporta una determinada función genérica de IOCTL.

**Kilobyte**
Unidad de medida equivalente a $2^{10}$ bytes. Es decir, 1,024 bytes.

**Macintosh**
Computadora desarrollada por la Empresa *Apple Computer*, la cual sentó precedentes en cuanto a ambientes de cómputo amigables. Su Filosofía es actualmente imitada por sistemas competidores.

**MASM**
Ensamblador de código de la empresa Microsoft, que inicialmente era embarcado con su Sistema Operativo DOS.

**Media Check**
Comando 1 en la interfaz definida por DOS para la comunicación con Controladores de dispositivos. Se aplica solo a dispositivos de bloque, como unidades de disco duro o flexible, unidades lectoras de disco compacto, etcétera. Su función es la de avisar al Sistema operativo si el dispositivo ha cambiado desde la última llamada. Su utilidad se manifiesta principalmente con las unidades de disco flexible, pues indica a DOS que se removió el último disco en usarse y se insertó otro. Típicamente, DOS enviaría en ese caso el comando 2, Get BPB, para conocer las características físicas y lógicas del nuevo dispositivo.

**Megabyte**
Unidad de medida equivalente a $2^{20}$ bytes. Es decir, 1'048,576 bytes.

**Memoria principal**
En una microcomputadora, es la memoria RAM que se localiza entre 0 y 1 MegaByte. Su sinónimo es Memoria convencional.

**Memoria secundaria**
Es la memoria no volátil, que es por lo normal más económica que la memoria RAM. Ejemplos son la memoria de disco duro, cinta de respaldo, etcétera.

**MFM**
Siglas de *Modified Frequency Modulation* (Modulación en Frecuencia modificada). Es un método tradicional de formateo de datos en un disco duro. Esta técnica implica la variación de amplitud y frecuencia de la señal que se aplica a un disco duro para leer o escribir información.

**Microprocesador**
es la Unidad central de procesamiento de una computadora.

**Microsoft**
Empresa de Computación Fundada en Seattle, Estados Unidos por William Gates. Creadora del Sistema Operativo DOS, que es el de mayor uso en microcomputadoras a nivel mundial en la actualidad.

**Modo de Direccionamiento Protegido**
Uno de dos modos de direccionamiento en el procesador 80286, y de tres en el 80386/80486. En este modo, los procesadores *80x86* usan todas las líneas de direccionamiento. Éste permite direccionar hasta 16 MegaBytes de memoria física en un procesador 80286 y hasta 4 GigaBytes en un 80386/80486. El manejo de memoria interna realizado por el 80286 le permite manejar 1 GigaByte adicional de memoria virtual en modo protegido, y al 80386 hasta 64 TeraBytes. Las direcciones en modo protegido son especificadas en formato Selector:Offset.

**Modo de Direccionamiento Real**
Uno de dos modos de direccionamiento en el procesador 80286, y de tres en el 80386/80486. En este modo todos los procesadores usan solo 20 líneas para direccionamiento, permitiendo así referenciar hasta un MegaByte de memoria física. El Modo Real no soporta Direccionamiento de memoria virtual. Las direcciones en este modo se especifican en formato Segmento:Offset.

**Mouse, vease Ratón**

**Multics**
Sistema operativo desarrollado a fines de la década de los sesentas en Estados Unidos. Se considera predecesor del Sistema operativo UNIX, aunque tienen pocas características comúnes debido a que Multics inicia como un Sistema operativo mucho más ambicioso.

**Nondestructive Input**
Comando 5 de la interfaz DOS a Controlador de dispositivos. Su objetivo es indagar al Controlador de un dispositivo para averiguar si tiene algún dato en su buffer de lectura interno esperando para ser leído. Típicamente este comando es seguido por un comando *Input*.

**Offset, vease Corrimiento**

**OS/2**
Sistema operativo que en el momento de su concepción (en 1985) fue considerado el sucesor de DOS. Su construcción fue iniciada por Microsoft e IBM, aunque las últimas versiones son propiedad de IBM. Tuvo otros nombres antes del actual, entre ellos DOS 5.

**Output**
Comando 8 de la interfaz DOS a Controlador de dispositivos. Indica al Controlador que escriba el contenido de un Buffer de datos al dispositivo.

**Output Flush**
Comando 11 de la interfaz DOS a Controlador de dispositivos. Usado para ordenar al Controlador de dispositivos que avise al dispositivo la necesidad de descartar cualquier dato aún esperando para ser escrito.

**Output Status**
Comando 10 de la interfaz DOS a Controlador de dispositivos. Este comando provoca que el Controlador cheque su dispositivo controlado para averiguar si está listo para recibir información. Este comando es por lo normal emitido antes de hacer una llamada al comando *Output*. Obviamente, debe siempre retornar un código de error cuando se trate de un dispositivo de solo lectura.

**Output Til Busy**
Comando 16 de la interfaz DOS a Controlador de dispositivos. Comando válido solo para dispositivos de tipo caracter. Demuestra su utilidad en dispositivos como impresoras que tienen un buffer para recibir datos. En vez de enviar un número pequeño de caracteres a la vez, se envía hasta llenar el buffer de datos y así se ahorra tiempo y llamadas.

**Output with Verify**
Comando 9 de la interfaz DOS a Controlador de dispositivos. Similar al comando Output, pero con una característica adicional: Cuando el switch del Sistema operativo VERIFY tiene un valor de ON, el controlador leerá los datos después de cada escritura. Este comando es valioso cuando se requiere manejar datos cuya integridad es de extrema importancia.

**Palabra, vease Word**

**Phoenix Technologies**
Compañía Norteamericana fundada en 1979 por Neil Colvin. Inició como proveedor de utilerías para desarrolladores pero es mejor conocida por su ROM BIOS para PC y compatibles, el cual salió a la venta en Mayo de 1984. Este producto resultó clave para hacer a las computadoras PC de otros proveedores compatibles 100% con las de IBM, dando así un gran impulso al mercado de microcomputadoras.

**Pista (Track)**
Segmento de un disco duro o flexible que es circular y por tanto puede ser accesado por la cabeza de Entrada/Salida durante una vuelta completa de la unidad giratoria.

**PSP**
Siglas de Program Segment Prefix (Prefijo de segmento del programa). Es un bloque de datos de 256 bytes de tamaño que DOS construye para cada

programa que ejecuta. Contiene información que el programa requiere para correr, además de información que DOS necesita mantener. Cada programa tipo EXE contiene su propio PSP al pricipio del archivo y DOS lo utiliza para construir el de memoria, mientras que cada programa tipo COM obtiene su PSP totalmente creado por DOS.

**RAM**

Siglas de *Random Access Memory* (Memoria de acceso aleatorio). Es la memoria volátil de una computadora. Es decir, aquella que pierde su contenido tan pronto como el equipo pierde su alimentación. Es más veloz pero también más cara que otros tipos de memoria.

**Ratón**

Dispositivo apuntador usado primordialmente para facilitar la interfaz a usuario en ambientes de cómputo gráficos.

**Removable Media**

Comando 15 de la interfaz DOS a Controlador de dispositivos. Es solo válido para dispositivos de tipo Bloque, como discos duros o flexibles. Este comando interroga al Controlador para saber si el dispositivo controlado contiene medios removibles. Para una unidad de disco duro la respuesta sería negativa, pero para una unidad de disco flexible sería positiva.

**Request Header**

El *Request Header* (Encabezado de Solicitud) es una estructura de datos que contiene información y espacio para comunicar un comando y su resultado entre DOS (o el programa que lo invoque) y un Controlador de dispositivos. Su estructura varía dependiendo del comando que se esté invocando.

**ROM**

Siglas de *Read Only Memory* (Memoria de solo lectura). Es aquella memoria cuyo contenido está permanentemente grabado, y que no cambia a pesar de no tener alimentación de corriente. En una microcomputadora se usa para grabar el código y los datos del BIOS.

**Sector**

Una de las áreas en que se divide una Pista de disco. Su tamaño es variable, aunque para DOS casi siempre se considera de 512 bytes.

**Segmento**

Un conjunto de direcciones de memoria contiguas. En el modo real de una microcomputadora este conjunto mide 64 KiloBytes. En modo protegido, un programa puede referenciar un segmento de cualquier tamaño.

**Set Logical Device**

Comando 24 de la interfaz DOS a Controlador de dispositivos. Comando disponible solo para dispositivos de bloque corriendo bajo la versión 3.2 o posterior de DOS. Este comando permite a los usuarios especificar múltiples letras de unidad para una sola unidad física. Esta asignación múltiple se logra a través del Controlador de dispositivos DRIVER.SYS, incluido en el Sistema operativo.

**Sistema operativo**
Conjunto del código usado por una computadora para administrar sus recursos e interactuar con su medio ambiente a través de sus dispositivos periféricos.

**Software**
Término utilizado para designar todos los componentes no Hardware de un equipo de cómputo. Éste incluye al *Sistema Operativo* y a los programas del usuario.

**Tarjeta controladora**
Un dispositivo de hardware consistente en una tableta de circuitos que puede ser insertada en una de las ranuras de expansión de una microcomputadora para expandir sus capacidades de Entrada/Salida, Almacenamiento o Procesamiento.

**TLIB**
Programa de Borland usado para administrar archivos de librería, los cuales contienen código objeto de funciones. El programa ligador puede tomar código directamente desde estos archivos librería.

**TLINK**
Programa ligador de código objeto de Borland cuya función es crear un programa ejecutable final a partir de las rutinas en código objeto que recibe como parámetros.

**Track, vease Pista**

**UNIX**
Sistema operativo surgido en 1970 de Bell Laboratories. Inicialmente fue desarrollado para un equipo DEC PDP-7, de Digital Equipment Corporation. En 1971 parte del sistema fue reescrita en Lenguaje C. Desde enctonces se han desarrollado versiones para gran cantidad de plataformas de cómputo. Es uno de los Sistemas operativos de más uso a nivel mundial.

**Windows**
Ambiente operativo de tipo gráfico que constituye el enfoque de la empresa Microsoft a la computación personal amigable. Inspirado en otros sistemas semejantes, como Elisa de Xerox o Macintosh de Apple Computers. Su tendencia es a convertirse en un Sistema Operativo para equipos personales en su siguiente versión (Windows 95). También da origen al Sistema Operativo para plataformas de Alto rendimiento Windows NT Server/Workstation. Existen otras tendencias, que están aún en desarrollo.

**Word (Palabra)**
En una microcomputadora, una palabra equivale a 2 bytes. Es decir, 16 bits.

**Write Precompensation (Precompensación de escritura).**
Un procedimiento donde la corriente aplicada a la cabeza de Entrada/Salida de una unidad de disco duro o flexible es variada dependiendo de la posición de la cabeza (Entre las pistas externas y las internas), y cuyo objetivo es mantener el nivel de la señal constante. Dicho parámetro frecuentemente se da como un número de cilindro, indicando el cilindro en el cual dará inicio el proceso de Precompensación.

**Z-80**

**Microprocesador de 8 bits producido por la empresa *Zilog* que fue uno de los pioneros en el campo de los Sistemas de cómputo personales. La empresa Intel lo tomó como inspiración para el desarrollo de su microprocesador 8080 de 8 bits, el cual es el antecedente de los actuales microprocesadores de la familia 80x86.**

# Bibliografía.

1. **Advanced MS-DOS Programming (Second Edition).**
**Duncan, Ray.**
**Microsoft Press.**

2. **Writing Device Drivers for SCO UNIX.**
**Kettler, Peter. Statler, Steve.**
**Addison Wesley.**

3. **CP/M and the Personal Computer.**
**Dwyer, Thomas A. Critchfield, Margot.**
**Addison Weisey.**

4. **Writing DOS Device Drivers in C.**
**Adams, Phillip M. Tondo, Clovis L.**
**Prentice Hall.**

5. **Writing MS-DOS Device Drivers.**
**Lai, Robert S.**
**Addison Wesley.**

6. **System BIOS for IBM PC/XT/AT Computers and Compatibles**
**Phoenix Technical Reference Series.**
**Addison Wesley.**

7. **Undocumented DOS.**
**Schulman, Andrew. Brown, Ralph. Maxey, David. Michels, Raymond. Kyle, Jim.**
**Addison Wesley.**

8. **Object Lifecycles, Modeling the world in States**
Shlaer, Sally. Mellor, Stephen J.
Yourdon Press Computing Series.

9. **Ingenieria de Software**
Fairley, Richard.
Prentice Hall.

10. **Writing a UNIX Device Driver**
Egan, Janet I. Teixeira Thomas J.
John Wiley & Sons.