CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

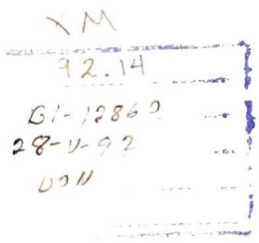
SECCION COMPUTACION

"SISTEMA OPERATIVO XINIX CON MANEJADOR DE DISCO DURO E IMPRESORA"

Tesis que presenta el Ing. David Leija Díaz para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA con opción en COMPUTACION.

Trabajo dirigido por el M. en C. Jorge Buenabad Chávez.

México D.F., abril 1992.


NM
92.14
61-12862
28-V-92
UNM

A mi esposa Nayma

A mis hijas Yael y Marcela

A mis padres Ignacia y David

A mis hermanos Melba y Javier

A mis amigos y compañeros

AGRADECIMIENTOS

Al M. en C. Jorge Buenabad por su apoyo y paciencia en la dirección y desarrolo de este trabajo.

Al Dr. Sergio Chapa por sus valiosos comentarios y por el tiempo dedicado a la revisión de este trabajo, de la misma manera, al M. en C. Andrés Vega por su apoyo.

A la M. en C. Ruth Delgado por su ayuda y comentarios.

A Petróleos Mexicanos por esta valiosa oportunidad de superarme profesionalmente. En particular, por el apoyo que me brindaron los siguientes amigos y compañeros de trabajo:

Ing. Héctor Palafox Rayón
Ing. Pagdo Mayo de la Vega
Ing. Antonio C. Limón

En especial a los Ingenieros:

Luis Madrigal Ugalde,
Guillermo Amaya García y al
Dr. Manuel Guzmán Rentería,

quienes planearon e hicieron posible la realización de esta Maestría PEMEX-CINVESTAV.

A la Sección de Computación, que con sus recursos hizo posible éste trabajo.

Al primer grupo seleccionado a esta maestría, formado por:

Ing. Hermilo Cruz Rivera
Ing. Alberto Zárate Ortega
Ing. Fernando Ramos Macías
Ing. Carlos Barajas Llerenas

del cual tengo el honor de pertenecer y agradezco el sentido de cooperación y apoyo que tenemos.

INDICE GENERAL

CAPITULO 4. MANEJADOR DE DISCO DURO PARA XINIX

CAPITULO 5. IMPLEMENTACION DEL SISTEMA DE ARCHIVOS TIPO UNIX DE XINIX EN DISCO DURO

INTRODUCCION

Antecedentes.

El Sistema Operativo (SO) XINIX fué desarrollado como trabajo de Tesis [1] en la Sección de Computación del Centro de Investigación y de Estudios Avanzados del I.P.N. (CINVESTAV-IPN). XINIX puede ser visto como un medio ambiente para el desarrollo de aplicaciones concurrentes en las computadoras IBM PC-XT, AT y compatibles. XINIX se construyó en base a los SO's XINU y MINIX. De XINU se tomó el código fuente, en lenguaje C, que forma la estructura principal de XINIX, y de MINIX, también en lenguaje C, se tomó el código fuente de los manejadores de disco flexible y terminal que controlan el hardware de la PC. XINU original corre en computadoras LSI-11 de DEC. Otra versión de XINU, llamada XINU-PC, se desarrolló en la Universidad de Wisconsin (USA). MINIX también opera en microcomputadoras IBM PC-XT, AT y compatibles.

Otro trabajo de Tesis [2] desarrollado también en el CINVESTAV-IPN, está relacionado con la incorporación de un Sistema de Archivos (SA) tipo UNIX para XINIX. Con éste trabajo se mejoró el sistema original de XINIX, debido a que éste último emplea un SA plano y pequeño: no tiene la facilidad de manejar subdirectorios y el número máximo de archivos que tiene permitido manejar en un disco flexible es 28.

Objetivos.

Ahora bien, con el Nuevo SA de XINIX (NSAX) permite tener una cantidad de archivos limitada sólo por la capacidad de almacenamiento del disco flexible.

XINIX original cuenta con un SA basado en un sólo disco flexible, los cuales son lentos y tienen una capacidad de almacenamiento restringido a 360 KBytes. El NSAX dió a XINIX la capacidad de manejar dos discos flexibles. Con ésta Tesis, el NSAX se maneja en disco duro, con mayor velocidad y obviamente mayor capacidad de almacenamiento. Por otra parte, los usuarios de XINIX no tenían la facilidad de imprimir sus resultados. Ahora es posible hacerlo, utilizando los comandos lpr y prt.

Propósitos de los Primeros Trabajos.

Cabe mencionar que los SO's XINU y MINIX fueron diseñados para la enseñanza de SO's, y que XINIX también tiene este propósito: se ha utilizado en los cursos de Sistemas Operativos, Programación en Tiempo Real, Computación Distribuida y SO's Distribuidos en el CINVESTAV-IPN.

Descripción del trabajo.

Los dos primeros capítulos refieren al Sistema Operativo XINIX [1], considerando sus características, organización física y sus servicios.

En el tercero se detalla acerca de la organización física y lógica del disco flexible, con breve explicación del manejador del disco flexible del SO original XINIX.

En el capítulo 4 muestra la organización física y lógica del disco duro, cómo maneja una petición y las implicaciones que tuviese el SO original XINIX instalado en el mismo, en el caso de conservar su SA original.

En el capítulo 5 se explica la implementación de un SA en general, su estructura lógica, cómo instalar el SA en disco duro y se desriben dos comandos del disco duro.

El capítulo 6 explica la organizació física y lógica de una impresora de matriz de puntos. También se da una breve descripción del manejador de impresora de XINIX.

Conclusiones.

Los resultados de ésta Tesis, son los siguientes:

Al empezar el desarrollo de ésta tesis, primero se optó por utilizar el manejador de disco duro que utiliza el SO MINIX, sin embargo, los resultados obtenidos fueron poco favorables. La otra alternativa fué utilizar los servicios de lectura/escritura que ofrece la Interrupción 13H del BIOS, teniendo la ventaja de hacer transportable el "software" y de esta manera pueda aplicarse a cualquier PC compatible con IBM.

Por otra parte, la implementación del presente trabajo se llevó a cabo en dos tipos de discos: uno de 20 Mbytes con 614 cilindros, 17 sectores y 4 cabezas, y el otro disco de 40 Mbytes con los mismos parámetros anteriores excepto el número de sectores que ahora es de 31. En ambos tipos de disco se tienen respuestas de acceso lectura/escritura similares a las que se tienen cuando se opera en MS-DOS.

Cabe mencionar que estos discos tienen estructurada la FAT de manera diferente; el de 20 Mbytes con una FAT de 12 bits y el de 40 Mbytes es de 16 bits. Esto implica efectuar ciertos ajustes (el programa x2win.c verifica, al iniciar XINIX, el tipo de SA que tiene el SO MS-DOS y realiza lo apropiado) cuando se copia un archivo del SA de DOS al SA de XINIX. En este caso particular las respuestas de lectura son lentas cuando se copian archivos mayores a 40 Kbytes. El motivo principal de este retardo se debe a que es necesario rastrear la tabla completa de la FAT (o en su defecto hasta encontrar el fin de archivo) y para hacer esto se utilizan **n** lecturas a la FAT (de acuerdo al tamaño del archivo) para comprobar la existencia del siguiente cluster y así sucesivamente hasta llegar al fin de archivo.

El programa que construye el SA de XINIX en el diso duro soporta particiones no mayores a 24552 bloques (1 bloque=1024 bytes) equivalente a 25 Mbytes.

Ahora bien, respecto a la impresora, cabe señalar que el comando lpr imprime textos normalmente, caracter por caracter incluyendo espacios y tabuladores. Su aplicación es equivalente al comando TYPE de DOS cuando es utilizado con direccionamiento de salida a la impresora.

El comando prt tiene salida formateada, en cada hoja va impreso el nombre del archivo que se está imprimiendo así como la fecha, hora y número de página. La implementación de éste comando se basó en la interrupción 17H del BIOS.

Referencias en Bibiolgrafía:

[1] Tesis, Jorge Buenabad 1989.
[2] Tesis, Ruth Delgado 1990.

CAPITULO 1

CONCEPTOS DE SISTEMAS OPERATIVOS

" Un SO es una colección organizada de programas y datos que están específicamente diseñados para la gestión de los recursos de un Sistema de computación con el propósito de facilitar la creación de programas de computación y controlar su ejecución sobre el Sistema. " [3].

Sayers define a un SO como:
" Un conjunto de programas y rutinas las cuales guían a una computadora en la ejecución de sus tareas y asiste a los programadores con ciertas funciones de soporte. " [4].

La definición de "American National Standard" es:
" El software el cual controla la ejecución de programas de computadora, y el cual provee planificación, depuración, control de E/S, contabilidad, compilación, asignamiento del almacenamiento, manejo de datos y servicios afines. "

Los componentes tangibles de una computadora y los programas constituyen un sistema de computación, juntos ofrecen utilidad a los usuarios. Existen básicamente dos tipos de programas: programas del sistema, que controlan los componentes físicos de la computadora o bien, ayudan a desarrollar más programas; y programas de aplicación desarrollados por los usuarios. La Figura 1.1 muestra los componentes básicos de un sistema de computación.

El programa de sistema más fundamental es el SO, el cual proporciona un medio común en el cual los otros programas de sistema y aplicación pueden operar; así como los mecanismos para comunicarse con los dispositivos periféricos.

Los SO's se desarrollaron inicialmente como respuesta a una necesidad: maximizar la utilización de la Unidad Central de Proceso (UCP) y de los dispositivos periféricos. Fué necesario automatizar el flujo de trabajo, poner las decisiones del manejo de los recursos del Sistema en la escala de tiempo de la computadora en lugar de la de un operador humano. Con esta automatización el SO fué también conocido como Sistema de Procesamiento en Lote (o Batch). El procesamiento en lote consistía en grabar los trabajos de los usuarios en una cinta de entrada a ser procesada por el SO. El SO procesaba los trabajos secuencialmente, sin intervención de un operador humano, hasta terminar con el último trabajo. Si algún trabajo imprimía resultados, éstos se grababan temporalmente en una cinta (de salida) que era procesada por una computadora pequeña dedicada a realizar la impresión.

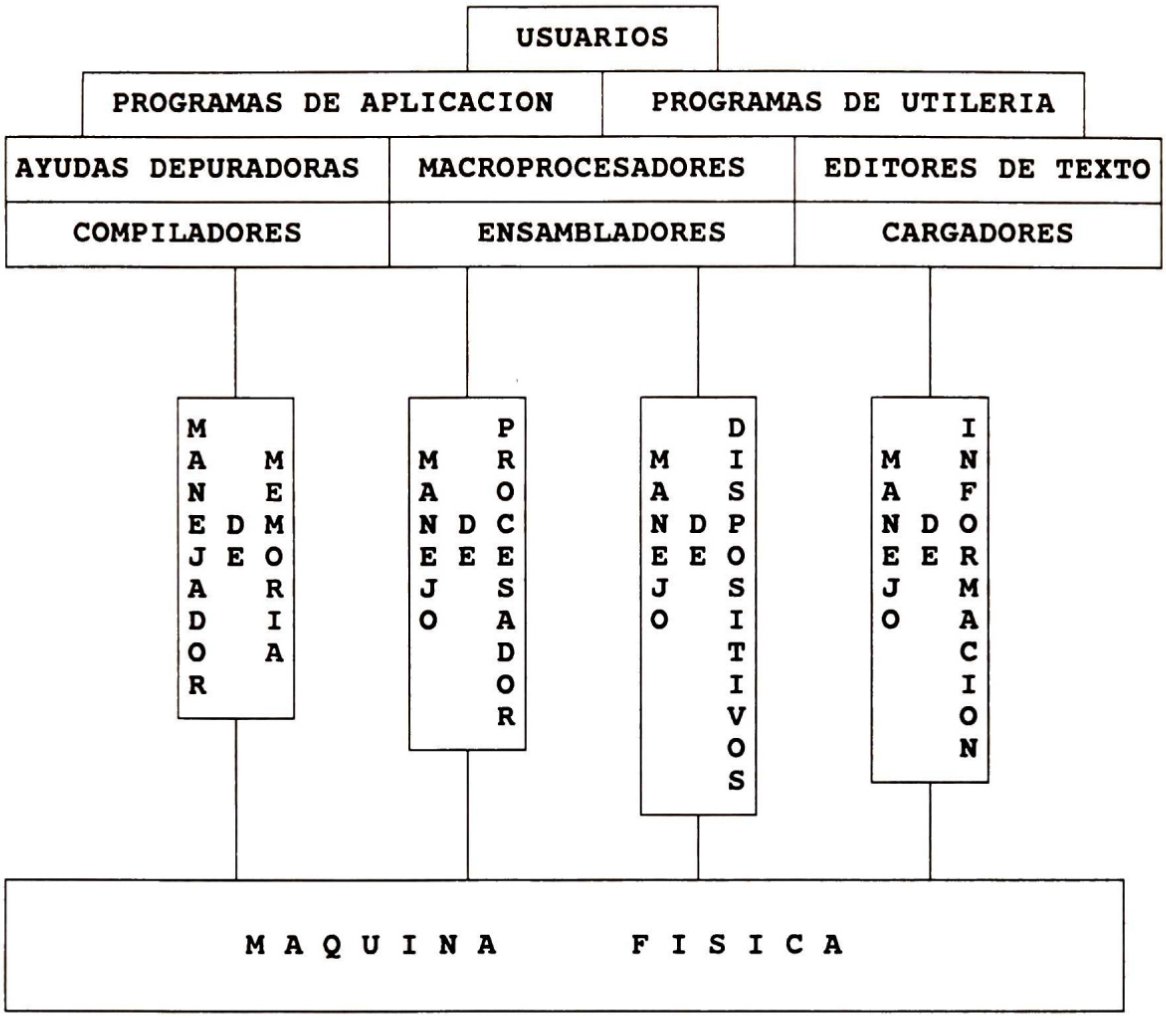


FIGURA 1.1.- RELACION DEL SO CON EL HARDWARE DE LA COMPUTADORA.

De esta manera la computadora principal procesaba trabajos de usuario la mayor parte del tiempo). Sin embargo, en cierto tipo de aplicaciones la UCP aún permanecía ociosa. Aplicaciones que procesaban muchos datos, y eventualmente los transferían de la memoria principal a dispositivos externos, o viceversa, utilizaban la UCP en controlar y checar que tales dispositivos terminasen las operaciones de entrada/salida (E/S) (Esto se debía principalmente al hecho de que la UCP es considerablemente más rápida que los dispositivos de E/S).

El procesamieto multitarea fué la respuesta a este uso improductivo de la UCP: toda tarea que realizara operaciones de E/S era suspendida hasta que tales operaciones terminasen, mientras tanto, otra tarea era ejecutada por la UCP.

Con el fin de proveer servicio de computación a más usuarios, el concepto multitarea se extendió entonces al de tiempo compartido, en el que las tareas o procesos se suspendían no sólo al solicitar operaciones de E/S, sino además periódicamente: dando intervalos de tiempo discontinuos a

todos los procesos hasta que éstos terminasen. Gracias a la gran velocidad de la UCP, los SO's de tiempo compartido hacen pensar a los usuarios que sus peticiones son atendidas continuamente.

Otro aspecto que se ha desarrollado constantemente en los SO's es el cualitativo. Los SO's han incluído muchas funciones y estructuras para facilitar a los usuarios el uso del sistema. Por ejemplo, en el SO OS/360 de IBM el usuario debe especificar el espacio a usar en disco a nivel de pistas, cilindros y sectores. SO's recientes manejan el espacio en disco por archivos referidos por el usuario con un nombre significativo. El SA se encarga de que el usuario vea el contenido de un archivo como una cadena continua de caracteres, aún y cuando éste se encuentre disperso en bloques en diferentes espacios del disco.

Así, un SO tiene dos funciones primordiales la primera es controlar y asignar los recursos a compartir entre varios procesos, la segunda es ofrecer a los usuarios una interfaz más conveniente y útil que la presentada por la máquina física.

Referencias en Bibliografía:

[3] Harry Katzan 1973.
[4] Sayers 1971.

CAPITULO 2

SISTEMA OPERATIVO XINIX

2.1. Características Generales.

XINIX es un SO de tiempo compartido para microcomputadoras IBM PC-XT, AT y compatibles, a las que pueden conectarse dos terminales a través de puertos serie RS-232.

El diseño de XINIX se basa en los SO's XINU [5] y MINIX [6]. De XINU se tomó el manejo, coordinación y comunicación entre procesos, el manejo de memoria y el sistema de archivos; de MINIX el manejador de disco flexible y el manejador de terminal.

La primera versión de XINIX , de aquí en adelante referido como XINIX solamente, permite solo utilizar el manejador de diskette "A" de la PC. El Sistema de Archivos (SA) de XINIX es el de XINU. Este SA es plano (no permite manejar subdirectorios), y el número de archivos que pueden existir simultáneamente en un diskette esta limitado a 28. XINIX tiene intérprete de comandos y es posible correr programas de manera interactiva. En XINIX se pueden ejecutar varios procesos concurrentemente, los cuales comparten el procesador y datos e instrucciones durante su ejecución. Esta concurrencia es la base del procesamiento multitaréa y del tiempo compartido en un SO. XINIX incluye ambas.

Las aplicaciones de los usuarios XINIX se desarrollan en el SO DOS, utilizando el compilador Turbo C de Borland; su ejecución empero es en XINIX, solicitándolo al intérprete de comandos una vez que XINIX se ha ejecutado como un programa de DOS.

En XINIX un 'proceso' es un programa en ejecución que puede estar en uno de los siguientes estados:

READY.- El proceso está listo para recibir el procesador.

SUSPEND.- El proceso se encuentra en el limbo, ni compite por el procesador ni espera por algún evento, sólo está latente, otro proceso debe sacarlo de su letargo.

WAITING.- El proceso espera por un evento; mientras tanto no compite por el procesador; el evento lo provoca otro proceso, y cuando ocurre, el proceso que esperaba por él pasa al estado READY.

RECEIVING.- El proceso espera por un evento en particular: un mensaje por parte de otro proceso; mientras espera no compite por el procesador; cuando recibe el mensaje pasa a READY.

SLEEPING.- El proceso espera a que transcurra un intervalo de tiempo para volver a pasar al estado READY.

CURRENT.- Es el estado del proceso que se ejecuta actualmente

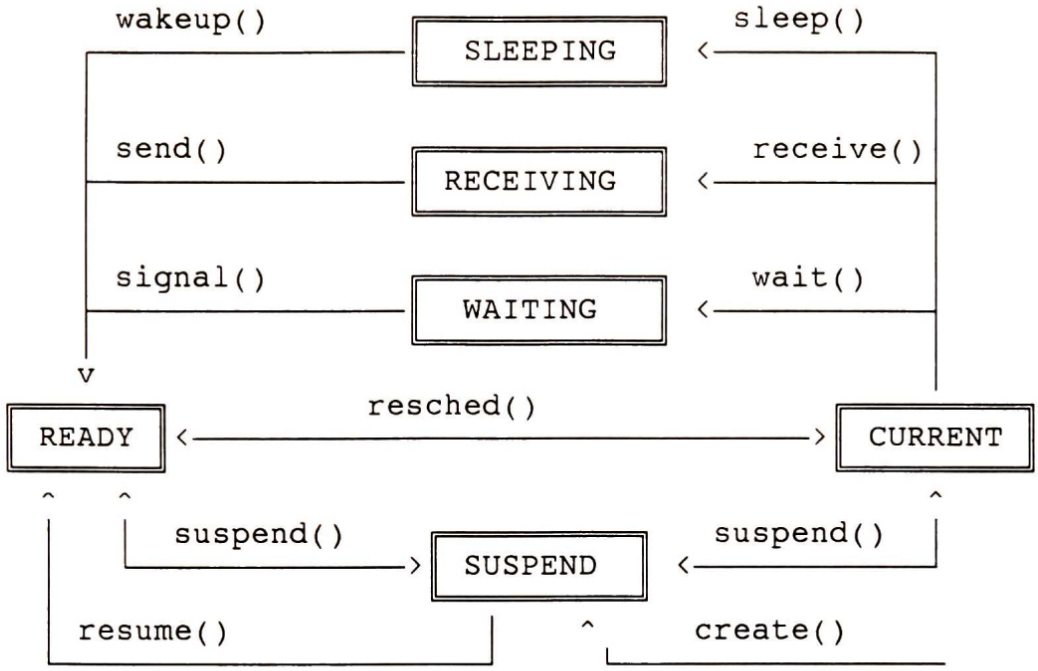


FIGURA 2.1.- ESTADOS DE LOS PROCESOS EN XINIX.

Los estados están indicados dentro de los cuadros, las flechas indican el sentido de las transiciones válidas, y los nombres seguidos de "()" son los servicios de XINIX que un proceso debe llamar para que ocurra una transición.

XINIX comparte o asigna el procesador entre los procesos READY. La asignación se hace de acuerdo a la prioridad de los procesos, y sólo se da al proceso con mayor prioridad, la prioridad varía de 0 a 32767, siendo 0 la menor. Cuando varios procesos tienen la prioridad actual más alta, cada uno recibe un intervalo de tiempo en el orden en que llegaron a competir por el procesador; al terminar su intervalo se colocan al final de los procesos con su misma prioridad, de esta manera, antes de recibir otro intervalo sus competidores han recibido el primero. Esta forma de asignación del procesador se llama "Round Robin", y tiene la característica de ignorar completamente a los procesos de menor prioridad: si existe un solo proceso con la prioridad más alta, sólo él se ejecutará.

Cuando no existe proceso alguno en estado "READY", el tiempo de procesador es consumído por el proceso "nulo", cuya prioridad es cero. Este proceso no hace nada, es un ciclo infinito.

2.2.- Organización Lógica.

La organización lógica se refiere a las capas (o componentes) de XINIX por el tipo de servicios que ofrece: manejo de memoria, manejo de procesos, sistema de archivos etc.. En la figura 2.2 se muestra la organización lógica de XINIX.

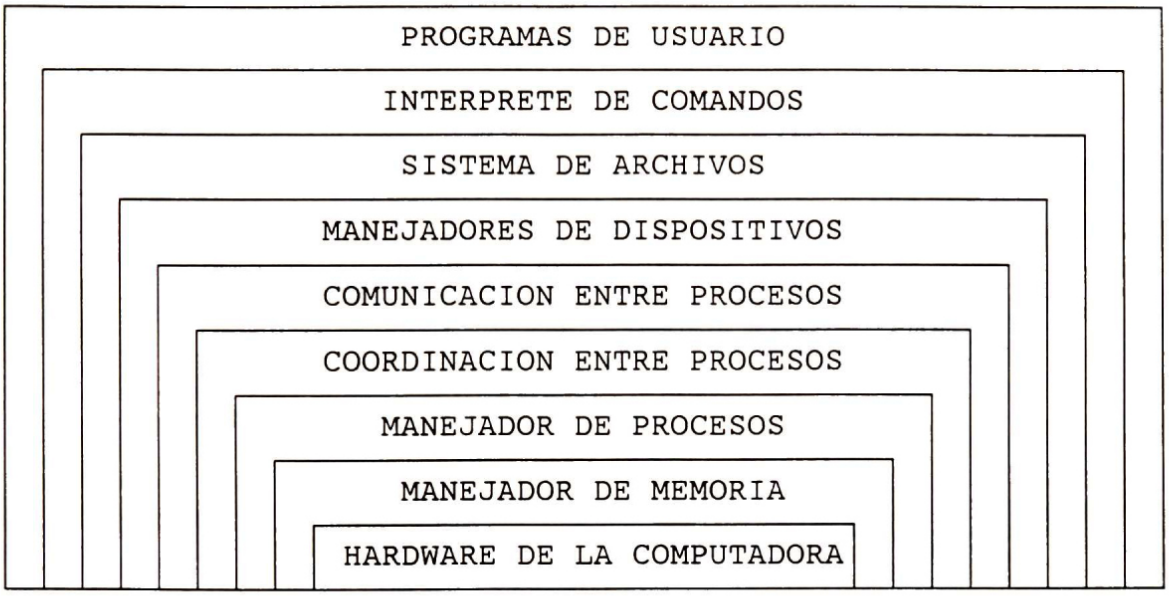


FIGURA 2.2.- LAS DIFERENTES CAPAS QUE FORMAN A XINIX.

Cada capa constituye u ofrece servicios para las capas superiores incluyendo los programas del usuario. Por ejemplo, el intérprete de comandos utiliza el servicio create(), en la capa "manejador de procesos", para crear un proceso a partir de un programa en disco. El programa a su vez puede, utilizando también a create(), crear procesos a partir de procedimientos encadenados con él.

Por otra parte, cada capa se forma de procedimientos y datos globales. Un servicio es un procedimiento en una capa, más no todos los procedimentos en ésta son servicios; por ejemplo, create() llama a newpid(), otro procedimiento del manejador de procesos, el cual obtiene la identificación que va
a asignar al proceso a crear. Newpid() busca una entrada libre en el arreglo global proctab[] (donde se encuentran los atributos de cada proceso existente en el sistema), y si la encuentra devuelve su número. La capa "coordinación de procesos" utiliza el arreglo semaph[] para implementar los semáforos. Para crear un semáforo, screate() llama a newsem(), el equivalente de newpid(), para buscar una entrada libre en semaph[]. Los procedimientos en otra capas también utilizan variables y arreglos globales para manejar dispositivos, bloques de memoria, archivos, almacenar los nombres de los comandos, etc..

En general, los procedimientos en cada capa realizan una función definida sobre un tipo de objeto en particular: un proceso, un semáforo, un archivo, un diskette, una terminal, etc. Por su parte, los objetos, sus atributos, son los campos que forman una entrada de un arreglo global en una capa; mientras que los valores en estos campos constituyen el "estado" actual del objeto: un proceso suspendido, un archivo abierto, un mensaje enviado, etc.; son estados que se detectan al preguntar por el valor de un campo atributo.

2.3.- Tipos de Servicios.

La utilidad de un SO depende de la cantidad y calidad de los servicios que ofrece a los programadores. Existen dos formas elementales de proveerlos: como llamados al sistema y como programas de sistema (o comandos). Los "llamados al sistema" son invocados por las aplicaciones de los usuarios durante su ejecución. Los "programas o comandos del sistema" los invoca un usuario al introducir su nombre por una terminal. A continuación se da una lista, clasificada por función, de los servicios que ofrece XINIX (las que son llamadas al sistema son precedidos por "+", comandos del sistema por "*"):

1) Servicios para el control de procesos.
   - Carga y ejecución de programas
     +* exec programa
     +* create programa
     +* resume programa
   - Terminación o concelación de un proceso.
     +* kill id-proceso
   - Obtención y modificación de los atributos de un proceso.
     +* chprio id-process newprio
      * get_priority id-proceso
      * get_status id-proceso
      * getpid
   - Suspensión temporal de un proceso.
     +* suspend id-proceso
     +* sleep n
   - Manifestar un evento en el sistema.
     +* send id-proceso mensaje
     +* signal id-semáforo
   - Suspensión de un proceso hasta que ocurra un evento particular.
      * wait id-semáforo
      * receive

2) Servicios para el manejo de archivos.
      * open id-dispositivo, nom-arch, modo
     +* close id-archivo
      * read id-archivo, buffer, num-chars
      * write id-archivo, buffer, num-chars
      * seek id-archivo, buffer, num-chars
     + cp archivo1 archivo2 (copia de archivos)
     + mv oldname newname (renombra archivos)
     + rm archivo (elimina archivos)
     + cat archivo (despliega contenido de archivo)
     + fmt (formatea un diskette)
     + vrf (verifica un diskette)
     + ls (lista un directorio)

3) Servicios para el manejo de memoria.
   * getmem nbytes
   * mkpool tamaño-buffer, num-bufs, areapoolblk
   * getbuff id-pool
   * freemem buffer
   * freebuf id-pool tamaño

4) Servicios para el manejo de dispositivos.
   * control dispositivo, función, arg1, arg2
   * breakt puerto
   * disablet puerto
   * initport puerto, vel, bpc, bp, p
   * enablet puerto
   * nobreakt puerto
   * getc dispositivo
   * putc dispositivo caracter

5) Servicios para mantener la información del sistema.
   + devs (lista los dispositivos)
   + who (lista la ident. de los usuarios en tty)
   + mem (muestra el estado de la memoria)
   + bpool (muestra el estado de las despensas de bloque)
   + ps (muestra el estado de los procesos)

6) Servicios utilería para el usuario.
   + cdxf dosfile xinixfile (copia un archivo desde un diskette con formato DOS a otro diskette con formato de XINIX).

2.4.- Manejadores de Dispositivos.

XINIX maneja concurrentemente los dispositivos de E/S de forma diferente al que realiza un SO monousuario como DOS. En DOS, cuando un programa lee datos del usuario a través de la terminal, el procesador se mantiene ocupado en un ciclo hasta que los datos son tecleados, lo que puede consumir un tiempo considerable si el usuario está distraído; esto sin embargo, no importa mucho pues dicho programa es el único que está ejecutándose. En XINIX esta situación se maneja en forma muy diferente, pues en éste puede haber más de un proceso requiriendo tiempo de procesador, por lo que éste no puede gastarse en esperar por un evento aleatorio como es el tecleo de caracteres. En XINIX, cuando un proceso lee datos de la terminal, éste es bloqueado (waiting) si no los hay, y el procesador se asigna al proceso en estado READY con la más alta prioridad. Cuando los datos se introducen, el proceso bloqueado pasa de nuevo al estado READY. Existen otras situaciones en las que DOS utiliza al procesador en checar la terminación de una operación con dispositivos, la lectura y escritura a disco es una de ellas. Para leer o escribir un sector de datos en el diskette, se programa el chip Floppy disk controller (FDC) y el chip Direct Memory Access (DMA):

se escriben en sus puertos asociados varias secuencias de bits que constituyen el comando (posteriormente se explicará con más detalle el funcionamiento de ambos chips). Al FDC se le indica la pista, el lado, el sector y el tipo de operación que va a efectuar (lectura/escritura). Mientras que al DMA se le indica la dirección de memoria, a partir de donde se ha de leer lo que se escribirá en el diskette, o donde se escribirá lo que se lea de diskette. La última secuencia de bits escrita en el FDC, según el comando, indica la operación de transferencia. DOS espera a que la operación termine: el procesador se utiliza en esta espera. El manejador de diskette de XINIX programa los chips FDC y DMA, pero en lugar de esperar el término de la operación ejecuta receive(): espera, sin competir por el procesador, por el mensaje que indica la terminación de la operación, el cual es enviado por el procedimiento que atiende la interrupción del FDC.

En DOS, este procedimiento tan solo cambia el valor de una variable, lo que indica a DOS que la operación ha terminado.

Referencias en Bibliografía:

[5] Comer, 1988.
[6] Tanenbaum, 1988.

CAPITULO 3

MANEJADOR DE DISCO (DISKETTE) ORIGINAL DE XINIX

3.1.- Organización Física de un Disco Flexible.

Para entender como están organizados los datos en un disco, consideraremos la estructura física del disco mismo, así como el mecanismo del manejador que lee y escribe datos en él. Empezaremos con los discos flexibles ("diskettes"), aunque los discos duros son similares, podríamos decir que tienen la misma estructura básica.

Dentro de la cubierta plástica de un disco flexible está un 'plato' circular hecho de un plástico resistente cubierto con un 'medio' magnético. Un manejador de disco flexible almacena datos sobre el disco escribiendo y leyendo patrones codificados magnéticamente, los cuales representan datos digitales. Ambos lados del diskette están cubiertos por éste medio magnético, así que se pueden utilizar ambas caras del diskette para almacenar datos. (La PC-IBM original salió al mercado con un manejador de diskette de una sola cabeza, por tal motivo, únicamente se utilizaba un lado del diskette, actualmente este tipo de manejador está fuera de uso).

La figura 3.1 muestra la estructura física de un disco flexible de 5 1/4 pulgadas de diámetro, 9 sectores por pista, y dos lados. Para un disco con esta estructura, un manejador de diskette incluye un motor que hace girar al diskette a una velocidad constante (después de un período de arranque de 2.5 segundos , aproximadamente), dos cabezas de lectura/escritura (L/E), una para cada lado del diskette, montados en los extremos paralelos de un brazo en forma de diapasón (el diskette gira entre los extremos paralelos), y los microchips "Floppy Disk Controller" (FDC) y "Direct Memory Access" (DMA) descritos en el Apéndice 4, que coordinan la transferencia de datos entre memoria y diskette.

Las cabezas de escritura/lectura pueden magnetizar el medio magnético del diskette para almacenar datos, o bien recuperarlos desde el diskette decodificando los patrones codificados magnéticamente en el medio del diskette.

3.1.1.- Qué es el Formateo Físico.

Para los discos flexibles, el proceso de formateo consiste de varios pasos. Primero, para transferir datos entre memoria y diskette el FDC recibe, además de la operación a realizar (lectura o escritura), la identificación de la pista, lado y sector inicial involucrados; el DMA recibe también el tipo de operación a realizar, junto con la dirección de memoria a partir de donde se encuentran los datos a transferir a disco, o a partir de donde se escribirán los datos que se lean del disco.

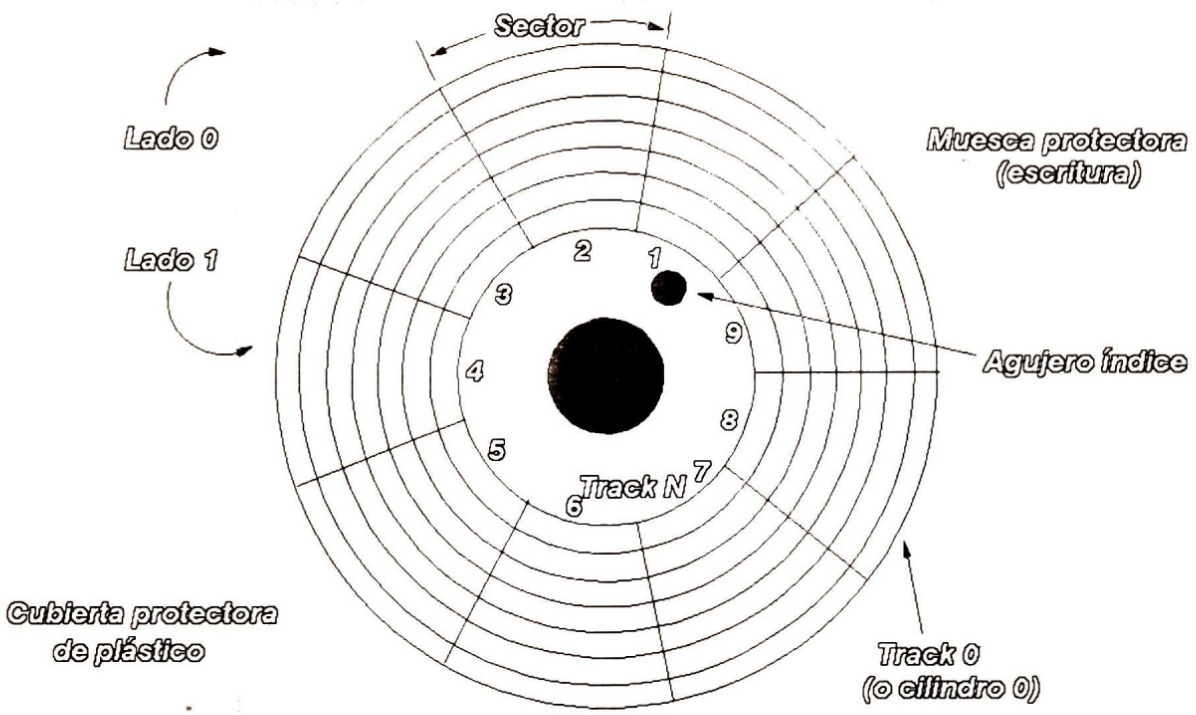


Figura 3.1

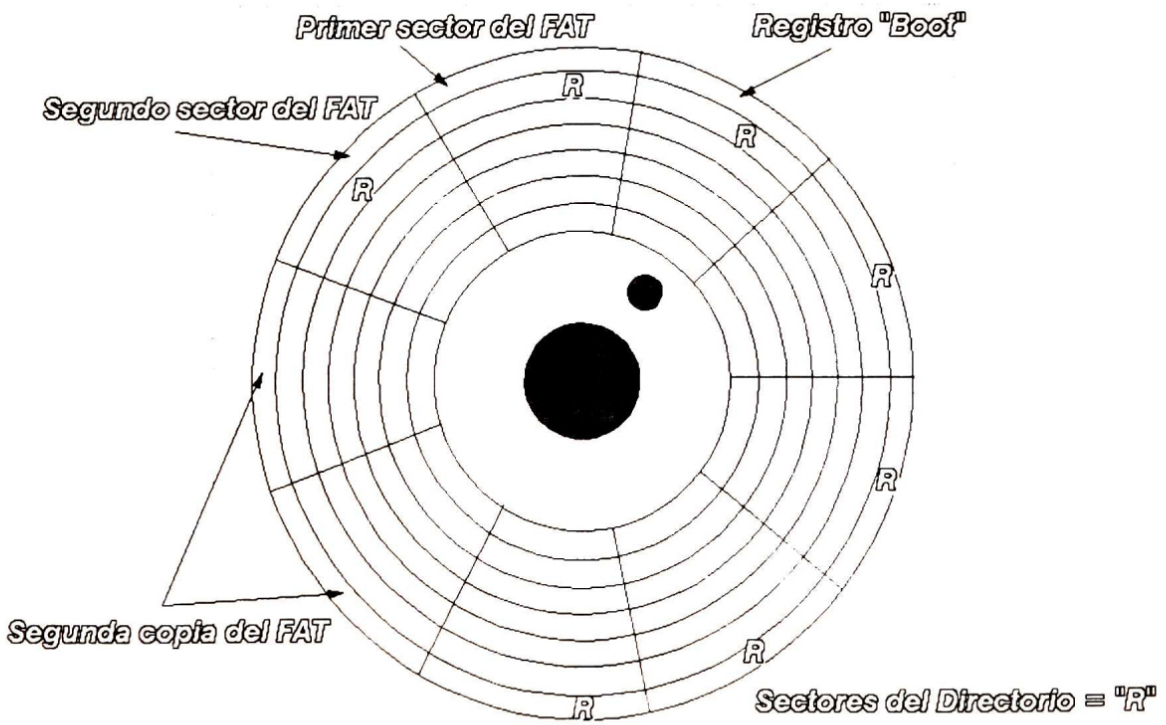


Figura 3.4

La transferencia de datos involucra cuatro etapas:

1) posicionamiento de las cabezas L/E en la pista involucrada.
2) selección del lado involucrado.
3) selección del sector inicial involucrado y,
4) lectura o escritura de los datos en el sector así identificado.

El FDC realiza la primera etapa desplazando la cabeza lectora, con un pulso eléctrico a la vez, tantas veces como el número de pista deseado (la pista cero no involucra desplazamiento, esto es cierto solo si antes de que cada posicionamiento las cabezas lectoras se muevan a la pista cero con el comando "recalibrate", tal como hacen los manejadores de disco de DOS y XINIX). La selección del lado involucrado se realiza activando eléctricamente aquél del cual se recibe su identificación (0 o 1). La selección de un sector, sin embargo, requiere la comparación del número de sector deseado con la identificación de cada sector en la pista lado ya seleccionados. Para esto, cada sector se compone en realidad de dos áreas: una en la que se almacena la identificación global del sector, ver figura 3.3, y otra donde se guardan propiamente los datos. La identificación global incluye el número de pista, de lado y de sector junto con la especificación del tamaño en bytes del sector (ver tabla 3.2).

Cuando un diskette no ha sido formateado físicamente no existen tales identificaciones globales de sector, no han sido impresas magnéticamente, y por lo tanto no se pueden leer ni escribir datos en el diskette. El formateo físico consiste, entonces, en grabar tales identificaciones globales de sector por pista-lado; es decir, una pista completa, un solo lado, a la vez. Para formatear una pista el FDC debe recibir el comando de formateo (fmt), el número de pista, la identificación del lado y el código ASCII de un caracter con el que rellenará toda el área de datos del sector; mientras que el DMA recibe un comando de escritura y la dirección, en memoria, donde se encuentra la identificacion global de todos los sectores a formatear en la pista, ver figura 3.4 (y tabla 3.1). Nótese que al formatear todas las pistas de un diskette estas, y sus sectores, se delinean magnéticamente. El número de tracks por superficie y el número de sectores por track son indicados por la versión del SO que se esté usando.

En el caso de los discos flexibles, DOS rellena cada sector con el caracter cuyo valor decimal es 246, o F6 en hexadecimal [7]. La operación de formateo identifica cada sector del disco sobre el medio de almacenamiento usando una entrada de 4 bytes que contiene la información mostrada en la figura 3.2. Esta entrada de 4 bytes se conoce como PLSN. La operación de formateo imprime magnéticamente un valor único en cada sector del disco, tal como se muestra en la Tabla 3.1.

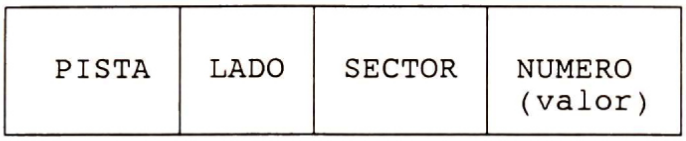


FIGURA 3.2.- SECTOR PLSN, 4 BYTES [7]

TABLA 3.1.- VALORES PLSN.

PLSN	Pista	Lado	Sector	Numérico
0012	0	0	1	2
0022	0	0	2	2
0032	0	0	3	2
0042	0	0	4	2
.	.	.	.	.
.	.	.	.	.
1012	1	0	1	2
1022	1	0	2	2
.	.	.	.	.
.	.	.	.	.
39192	39	1	9	2

Durante el servicio de formateo (ver figura 3.1) el FDC usa el agujero índice del diskette para determinar donde colocar el valor PLSN del sector inicial. Una vez realizado el formateo, los servicios de lectura/escritura de sectores puede llevarse a cabo.

Generalmente todo SO ofrece un comando para realizar el formato físico y lógico (que se trata más adelante) de un diskette. En DOS este comando se llama "FORMAT", en XINIX se llama "fmt". Por otra parte, es posible que un programa de usuario realice el formateo físico de un diskette, en cuyo caso existen dos alternativas: la primera consiste en programar directamente los chips FDC y DMA, lo cual es bastante complicado (ver una instancia de esta programación en la sección 3.1.2), la segunda alternativa consiste en utilizar el servicio 5, interrupción 13H, del ROM-BIOS (BIOS=Basic Input Output System), el cual utiliza un buffer que contiene 4 bytes CHRN de entrada por cada sector sobre el track, observe la figura 3.3. El valor N de PLSN

especifica el tamaño del sector del disco, la Tabla 3.2 describe su interpretación.

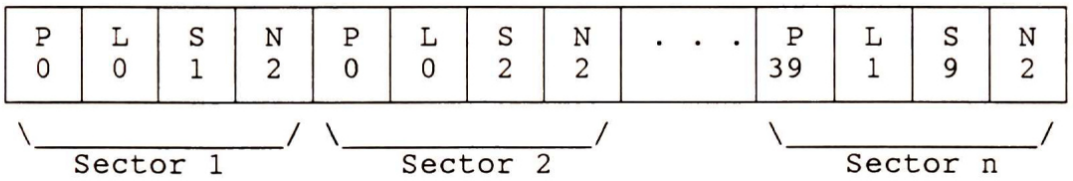


FIGURA 3.3.- CUATRO BYTES DE PLSN.

TABLA 3.2.- VALORES N DE PLSN.

Valor numérico	Tamaño del Setor (Bytes)
0	128
1	256
2	512
3	1024

3.1.2.- Programación Directa del Hardware que Controla un Diskette.

Las operaciones básicas que se pueden realizar con el hardware de un diskette son:

- SPECIFY
- SEEK
- RECALIBRATE
- FORMAT TRACK
- READ DATA
- WRITE DATA

Existen otras operaciones, todas se tratan en el Apéndice 4, donde también se describe el funcionamiento del hardware de un manejador de diskette.

A continuación se ejemplifica la programación directa de los microchips FDC y DMA, implementada en lenguaje C, para leer un sector de datos y formatear una pista:

```
/*-------------------------------------------------------
 * transfer -- El manejador está ahora en el cilindro
               indicado, lee o escribe un bloque
 *-------------------------------------------------------
 */

 LOCAL int transfer()
 {
     int r, s, op;

/* No intente transferir si el manejador está descali-
   brado o el motor está apagado */
 if (fp->fcalibration[fp->fdrive] == UNCALIBRATED) {
         fprintf(fp->fstdio, "transfer UNC\n" );
         return(ERR_TRANSFER);
 }
 if ((( Bios_Motor_Status >> (fp->fdrive + 4)) & 1)==0 )
 {
         fprintf (fp->fstdio, "transfer BMS\n" );
         return(ERR_TRANSFER);
 }
 /* Se realiza el comando enviando 9 bytes al chip
    controlador */
 op = (fp->fopcode == DISK_READ || fp->fopcode ==
      DISK_READ_TRK ?  FDC_READ: FDC_WRITE);
 fdc_out(op);
 fdc_out( (fp->fhead << 2) | fp->fdrive);
 fdc_out(fp->fcylinder);      /* se le dice al contro-
                                 lador qué cilindro */
 fdc_out(fp->fhead);          /* qué cabeza ?  */
 fdc_out(fp->fsector);        /* cual sector ? */
 fdc_out( (int) len[ SECTOR_SIZE/DIVISOR ]);  /* ta-
                                 maño del sector */
 fdc_out(NR_SECTORS);         /* qué tan grande es la
                                 pista*/
 fdc_out(GAP);                /* qué tan grande es le
                                 gap */
 fdc_out(DTL);                /* cual es la longitud
                                 del dato */

 if (need_reset)
     return(ERR_TRANSFER-20); /* aborta operación */

 /* El bloque espera por la interrupción del disco */
 if ( receive() == NOT_READY)
     return(NOT_READY);

 /* Obtiene el estado  del controlador y comprueba
   errores */
 r = fdc_results();
 if (r != OK)
     return(r);
```

```
  if ((fp->fresults[ST1]&BAD_SECTOR) || (fp-
        >fresults[ST2]&BAD_CYL))
      fp->fcalibration [fp->fdrive] = UNCALIBRATED;
  if (fp->fresults[ST1] & WRITE_PROTECT)
      return(ERR_WR_PROTECT);
  if ((fp->fresults[ST0]& ST0_BITS) != TRANS_ST0)
      return(ERR_TRANSFER-30);
  if (fp->fresults[ST1] | fp->fresults[ST2])
      return(ERR_TRANSFER-40);

  /* Compara el número actual de sectores transferi-
     dos con el número esperado */
  s = (fp->fresults[ST_CYL] - fp->fcylinder)* NR_HEADS *
     NR_SECTORS;
  s += (fp->fresults[ST_HEAD] - fp->fhead) * NR_SECTORS;
  s += (fp->fresults[ST_SEC] - fp->fsector);
  if (s * SECTOR_SIZE != fp->fcount)
      return(ERR_TRANSFER-50);
  return(OK);
}
```

```
/*-------------------------------------------------------
* formater -hace el comando de formateo sobre el mane-
           jador especificado.
*-------------------------------------------------------
*/
LOCAL formater()
{
     int r;
  /* No intente formatear si el manejador está descali-
     brado o el motor está apagado */

  if (fp->fcalibration[fp->fdrive] == UNCALIBRATED) {
        xwrite (CONSOLE, "formater UNC\n", 13);
        return(ERR_FORMATER);
  }
  if ((( Bios_Motor_Status >> (fp->fdrive + 4)) & 1)==0)
  {
        xwrite (CONSOLE, "formater BMS\n", 13);
        return(ERR_FORMATER);
  }

  /* Se realiza el comando enviando 6 bytes al chip
     controlador */
     fdc_out(FDC_FORMAT);
     fdc_out( (fp->fhead << 2) | fp->fdrive);
     fdc_out( (int) len[ SECTOR_SIZE/DIVISOR ]);  /* ta-
                                    maño del sector  */
     fdc_out( NR_SECTORS );     /* cuantos sectores en
                                    una pista*/
```

```
        fdc_out(GAP_FORMAT);        /* qué tan grande es el
                                       gap entre sectores */
        fdc_out(fp->fsector);       /* caracter de llenado para
                                       formateo */

        if (need_reset)
             return(ERR_FORMATER-20); /* aborta operación */

        /* Bloque esperando por interrupción del disco */

        if ( receive() == NOT_READY)
             return(NOT_READY);

        /* Obtiene el estatus del controlador y comprue-
           ba errores */

        r = fdc_results();
        if (r != OK)
             return(r);
        if ((fp->fresults[ST1]&BAD_SECTOR) || (fp-
           >fresults[ST2]&BAD_CYL))
             fp->fcalibration [fp->fdrive] = UNCALIBRATED;
        if (fp->fresults[ST1] & WRITE_PROTECT)
             return(ERR_WR_PROTECT);
        if ((fp->fresults[ST0]& ST0_BITS) != TRANS_ST0)
             return(ERR_FORMATER-30);
        if (fp->fresults[ST1] | fp->fresults[ST2])
             return(ERR_FORMATER-40);
        return(OK);
}
```

3.2.- Organización Lógica de un Disco Flexible.

3.2.1.- Qué es el Formateo Lógico.

Realizado el formateo físico, procede entonces el formateo lógico del diskette, que consiste en establecer las áreas de trabajo del diskette. Estas áreas de trabajo son usadas para controlar cómo y dónde deberán almacenarse los archivos. El formateo lógico de un disco, puede ser disco duro o disco flexible, consiste en grabar tres áreas diferentes de trabajo: el área o registro de "booteo", el área de información de control del sistema de archivos (ICSA), y el área donde se almacenan los datos de los archivos. Vea la figura 3.4. (Al encender una computadora personal, un procedimiento de inicialización del ROM-BIOS toma el control con el fin de determinar las características del hardware instalado, tales como capacidad de memoria,

tipo de monitor, etc.; realizado este chequeo, este procedimiento lee y pone en memoria el primer sector del diskette "A" o del disco duro, asumiendo que su contenido es el registro de booteo, y entonces le pasa el control: saltando a éste para ejecutarlo).

El registro de booteo (Vea la figura 3.5) es un programa encargado de leer el SO en disco, colocarlo en memoria y pasarle el control. Después de un período de inicialización todo SO corre un programa (que indistintamente llamaremos intérprete de comandos o "shell", y) cuya función es servir como interfaz entre el SO y el usuario.

INFORMACION DE FORMATEO DEL REGISTRO "BOOT"

Offset (dec)	(hex)	Tamaño	Contenido
0	00	3 Bytes	Short JMP al Código de "boot".
INFORMACION		DE	FORMATEO
3	03	8 Bytes	Nombre OEM y versión.
BIOS		PARAMETER	BLOCK
11	0B	1 Word	Bytes por sector.
13	0D	1 Byte	Sectores por cluster.
14	0E	1 Word	Número de sectores reservados.
16	10	1 Byte	Número de tablas del FAT.
17	11	1 Word	Número de entradas de directorios.
19	13	1 Word	Número de sectores logicos.
21	15	1 Byte	Byte descriptor del medio.
22	16	1 Word	Número se sectores del FAT.
24	18	1 Word	Sectores por track.
26	1A	1 Word	Número de cabezas.
28	1C	1 Word	Número de sectores ocultos.
30	1E	416 Bytes	Código del "boot".
446	1BE	16 Bytes	Información de la partición
462	1CE	50 Bytes	Resto del código del "boot".

FIGURA 3.5.- DESCRIPCION DEL REGISTRO "boot" [8].

Otro propósito del registro boot es el de servir como un registro de identificación (ID) de las características del diskette,tal como el número de tracks por lado y sectores por track.

El área ICSA contiene la información necesaria para administrar el espacio en disco por archivos. Para un

usuario, un archivo es un espacio en disco, contiguo e ilimitado (hasta donde la capacidad del disco lo permita), que es referido con un nombre significativo para el usuario. (Los archivos pueden contener datos, un programa fuente, etc.).

Los datos de un archivo pueden o no estar en sectores contiguos (ésto último ocurre después de borrar y crear archivos en un diskette), sin embargo, el sistema de archivos siempre presenta los archivos como si estuviesen contiguos. Así parte del área ICSA contiene los nombres de los archivos y sus atributos, y otra parte contiene la información de control necesaria para recuperar los datos del archivo secuencialmente. Esta información de control puede organizarse de muchas maneras (su organización es diferente en los SO's UNIX XINIX Y DOS), más el propósito es el mismo: ofrecer el mecanismo para recuperar secuencialmente los datos de un archivo. Para dar un bosquejo de cómo se utiliza esta información de control, en esta sección se ilustra su organización y uso en el SO DOS.

En DOS, el área de ICSA contiene el directorio y la Tabla de asignamientos de archivo (FILE ALLOCATION TABLE, FAT) [8]. (En DOS, parte de la información especial contenida en el registro de "boot", asiste al SO en determinar el formato físico del diskette y en localizar y leer los archivos almacenados en el diskette. Vea la figura 3.5, en ella se muestra con detalle el contenido del registro del "boot".

Inmediatamente siguiendo al registro de "boot", está la FAT (Hay dos tablas de este tipo, actualmente). Esta tabla es una área especial de trabajo reservada para mantener las ligas de las localidades físicas de los archivos almacenados en el disco. Esta área de trabajo se establece e inicializa durante la segunda, o fase del procedimiento lógico de formateo. Un ejemplo de cómo DOS organiza la FAT lo podemos ver en la figura 3.6.

Adyacente a la FAT está el directorio raíz del diskette ("root directory"). El "root directory" es un archivo especial que contiene entradas para cada archivo individual almacenado en el disco. Como con la FAT, el directorio es inicialmente ocupado con datos. Las entradas actuales en el directorio del diskette ocurren cada vez que los archivos son agregados al diskette. En la figura 3.7 se ilustra la forma en que MS-DOS maneja los nombres de directorios o archivos. Esta estructura se compone de 32 bytes, si por ejemplo, en un diskette de 9 sectores y doble densidad, tenemos 7 sectores asignados unicamente para el área de directorios y conociendo de antemano que el tamaño de cada sector es de 512 bytes, entonces sólo podemos tener 112 nombres de archivo (o subdirectorios en el directorio raíz. No hay que olvidar que esto es para el caso particular del diskette mencionado; el número de directorios o archivos varía de acuerdo al tipo de diskette y versión del SO usado.

En los discos duros depende de cómo se formateó el disco, y según sea el número de sectores que se asignen al área de directorios, se podrá decir cual es el número máximo de archivos o directorios que soporte el directorio raíz.

No obstante que el registro "boot" está siempre localizado en el primer sector, el tamaño y localidades resultantes del FAT y directorio varía de acuerdo al tipo de formato empleado en el diskette. Estas dos características de tamaño son parte de la información de formato del diskette y son grabados en el registro "boot". Usando esta información, DOS puede determinar donde se localiza la porción de los datos en el diskette.

La porción de datos es el área asignada para los archivos de trabajo, es decir, es el área que dispone el usuario para guardar su información.

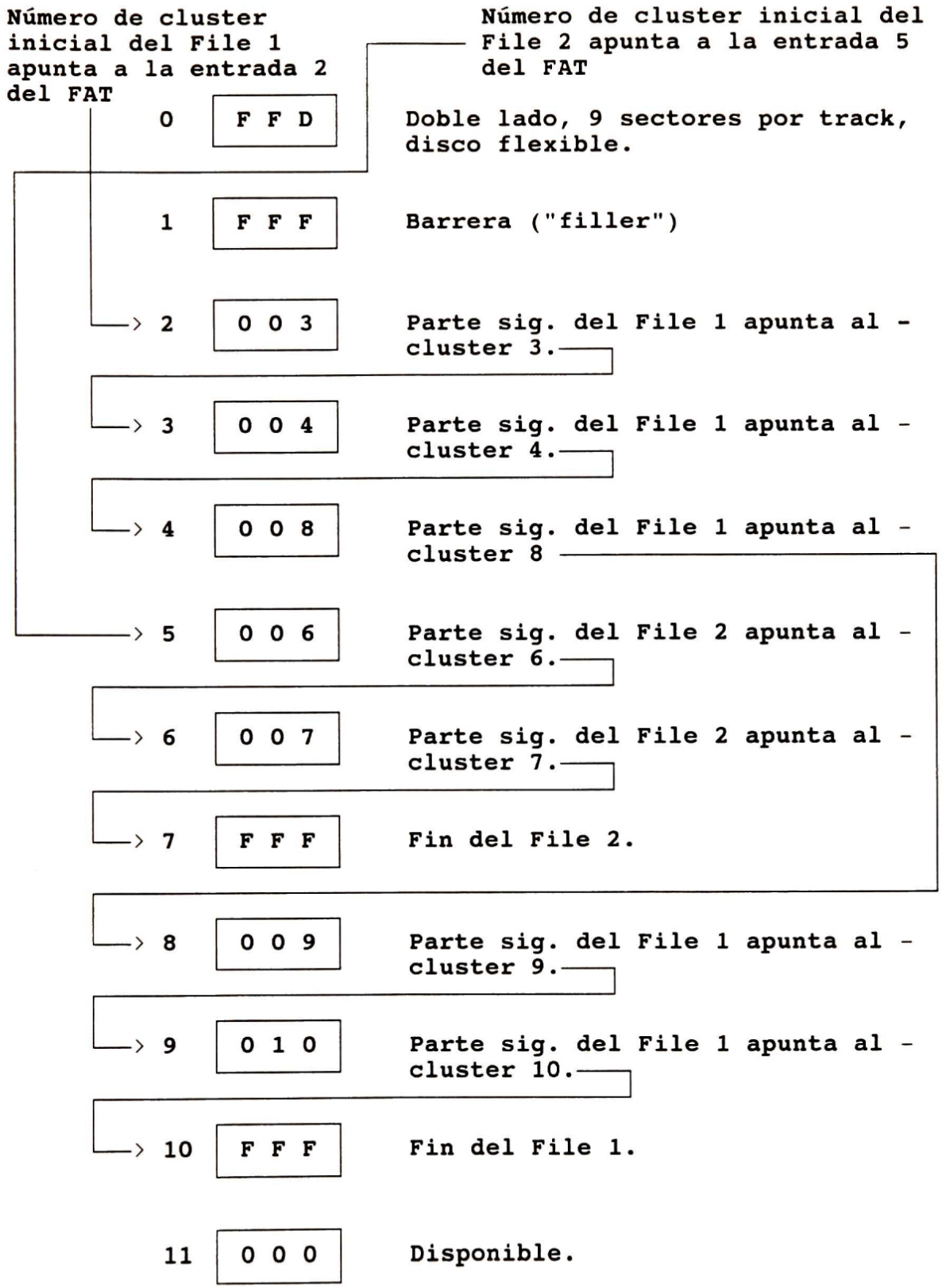


FIGURA 3.6.- SECUENCIA DE DOS ARCHIVOS ORGANIZADOS POR LA FAT.

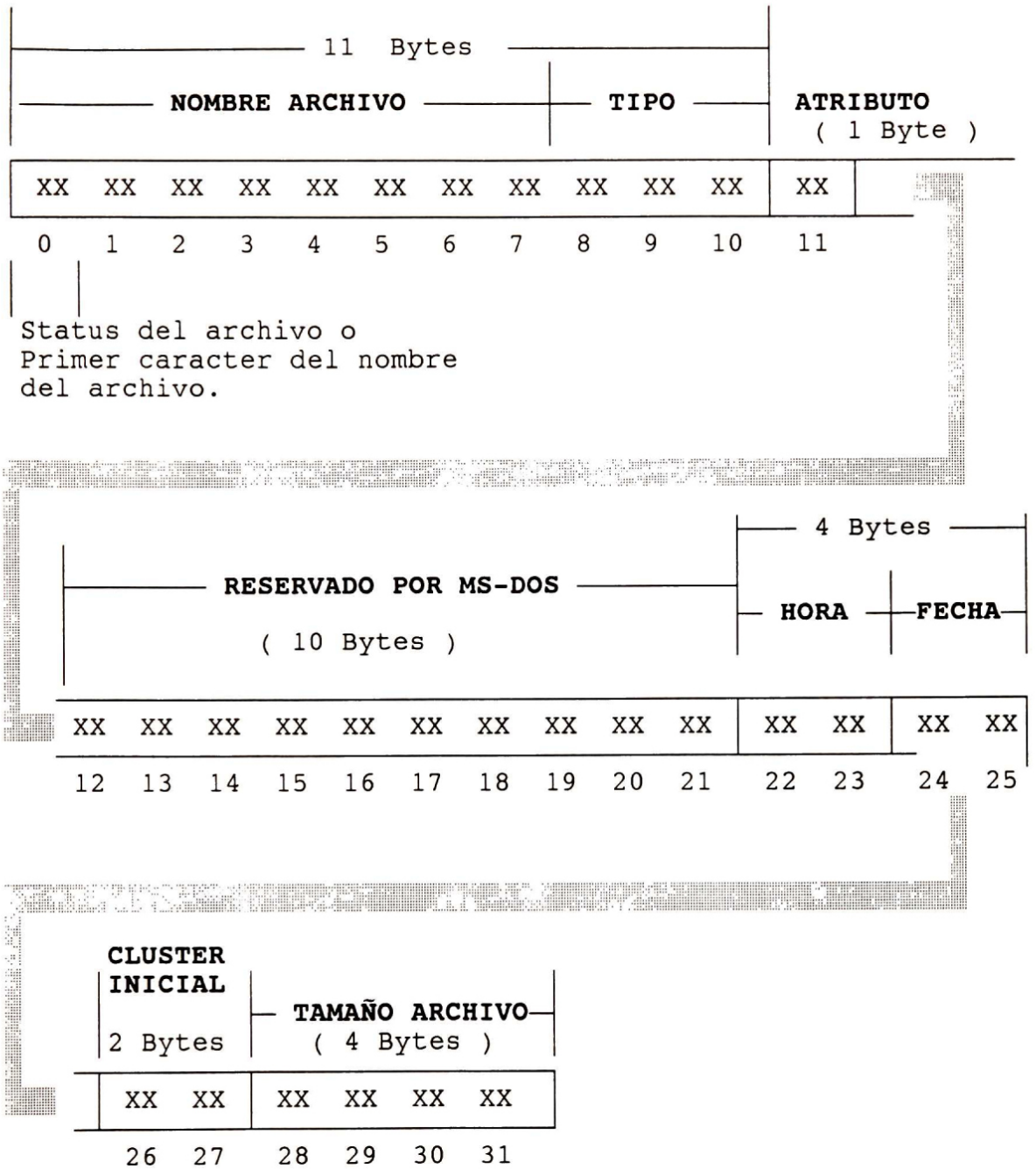


FIGURA 3.7.- COMPONENTES DE UN DIRECTORIO O ARCHIVO [8]

3.2.2.- Mapeo de un Número Lógico de Sector a Pista, Lado, Sector.

En DOS, las interrupciones 25H (37 decimal) y 26H (38 decimal), pueden ser usadas para leer y escribir sectores específicos de disco. Estos dos servicios de DOS ignoran la estructura lógica del disco y trabajan solamente con sectores individuales, no distingue las localidades de los

archivos, directorios, o al FAT [9].

Las Interrupciones 25H y 26H son similares a las correspondintes a los servicios del disco del ROM BIOS, excepto que los sectores son localizados por un método de numeración diferente. Con los servicios del ROM BIOS, los sectores son seleccionados por cilíndro, cabeza, sector, mientras que las Interrupciones 25H y 26H, los sectores son seleccionados por su numeración de sectores lógicos secuenciales.

La siguiente fórmula convierte el sistema de coordenadas tridimensional usado por el ROM BIOS a número de sectores lógicos usados por DOS:

```
Sector_Lógico = (Sector-1) + (Head*Sectores_x_track) +
                (Cilindro*Sectores_x_track*Num_Heads)
```

Y por otra parte, aquí tenemos las fórmulas para convertir números de sectores lógicos a coordenadas tridimensionales (Cilindro, Cabeza, Sector):

```
Sector = 1 + Sector_Lógico % Sectores_x_track
Head = (Sector_Lógico / Sectores_x_track) % Num_Heads
Cilindro = Sector_Lógico / (Sectores_x_track*Num_Heads)
```

No hay que olvidar que el ROM BIOS cuenta las cabezas y cilindros desde 'cero' pero los sectores desde 'uno'; los sectores lógicos de DOS están numerados desde 'cero'.

Las fórmulas del sistema de coordenadas tridimensionales son utilizadas en el manejador del disco flexible y de disco duro.

3.2.3.- Sectores de 512 Bytes o Múltiples, Cómo se Mapean.

La manera en que los datos se mapean en los discos flexibles y el disco duro es un resultado natural de la geometría de ellos mismos, es decir, el tamaño de los sectores del disco flexible y del disco duro son típicamente ambos de 512 Bytes (sectores físicos). En MS-DOS un diskette de dos caras, doble densidad (9 sectores), opera con bloques de 1024 Bytes, los cuales son llamados "clusters". Estos "clusters" varían de acuerdo al formato utilizado. El número de sectores en un cluster es siempre una potencia de dos; generalmente en discos duros tenemos clusters de 4 u 8 sectores. En el Capítulo 4 se dará una mejor explicación sobre el término "cluster".

Para XINIX original tenemos sectores de 512 bytes. Y con la versión de XINIX con el Sistema de archivos tipo UNIX, se puede manejar bloques de 1024 Bytes en disco flexible, así mismo se manejan en el disco duro.

Si se desea mayor información sobre la estructura física y lógica de los discos flexibles, el lector puede hacer referencia a las secciones 3.1, 3.1.1 y 3.2.1.

3.3.- Manejador Original de XINIX (archivo x2dskdrv.c).

3.3.1.- Cómo Acepta las Peticiones, su Estructura.

XINIX tiene una lista de peticiones para el disco flexible y ésta lista la controla el proceso disk_operator().

```
struct dreq  {    /* Nodo en la lista de solicitudes */
   int drfnum;    /* # de disco flexible: '0' o '1'  */
   DBADDR drdba; /* dirección del bloque del disco en
                    uso */
   int drpid; /* ident. del proceso que hace la solici-
                 tud */
   int drsizet; /* tamaño del bloque que se va transfe-
                   rir: (1)512,(2)1024,(3)2048,(4)4096*
   char *drbuff; /* direc. del buffer para lect/escri *
   char drop;  /* operación: READ/WRITE/SEEK */
   int drstat; /* status que regresa OK/SYSERR */
   int drstdio;/* dispositivo para enviar mensaje u ob-
                  tener opciones */
   struct dreq *drnext; /* apuntador al sig. nodo de la
                           lista de peticiones */
 }
```

Dicho proceso programa al FDC y al DMA. Cuando no hay peticiones a disco el proceso disk_operator() queda en estado SUSPEND; pasa al estado RECEIVE cuando espera que el FDC termine la operación actual; en SLEEP, cuando espera que un motor del diskette, apagado, alcance su velocidad de operación antes de realizar la transferencia; y en CURRENT o READY, cuando prepara y realiza la programación del FDC.

Cuando se hace una petición al disco flexible, los procesos preguntan si la lista de peticiones está vacía; si es así enlistan su solicitud y reanudan la ejecución de disk_operator() con el servicio resume(); de lo contrario, ponen su petición al final de la lista. Disk_operator() va desencolando y realizando las peticiones conforme llegan, es decir, este proceso tiene la política de planificación FIFO; la petición que llega primero es la que primero se atiende. Y cuando no hay peticiones, disk_opeartor() se suspende.

3.3.2.- Cómo Cambia el Tamaño del Sector Lógico.

El procedimiento dskstrt(), ejecutado por el proceso disk_operator(), es el que inicia una operación de E/S con los discos flexibles. La secuencia de operaciones que lleva a cabo es el siguiente:

1.- Programa la petición para al FDC.
2.- Comprueba si el número de "floppy" que efectúa la operación es el correcto ('0' o '1').
3.- Se obtiene el número de "floppy" actual y se guarda en la variable "driver". Por "default" se utiliza el "floppy" 0 (DISK0, dispositivo de XINIX).
4.- Se obtiene del campo **drsizet** de la estructura dreq, el tamaño del bloque que se va a transferir. Por default se tiene seleccionado que maneje bloques de 1024 bytes.
5.- Continúa el código, se procede a identificar el proceso, el número del drive, etc..

En el punto cuatro es donde se lleva a cabo el cambio del tamaño del sector lógico. En esta parte existen 4 opciones, son 4 macros que se muestran a continuación:

```
case 1: set_blk_512(driver);
case 2: set_blk_1024(driver);
case 3: set_blk_2048(driver);
case 4: set_blk_4096(driver);
```

Estas macros están definidas en el archivo xdisk.h, de la siguiente manera:

```
#define set_blk_512(ddev)  \
                dstab[devtab[(ddev)].dvminor].dsizeb = 512
#define set_blk_1024(ddev)  \
                dstab[devtab[(ddev)].dvminor].dsizeb = 1024
#define set_blk_2048(ddev)   \
                dstab[devtab[(ddev)].dvminor].dsizeb = 2048
#define set_blk_4096(ddev)    \
                dstab[devtab[(ddev)].dvminor].dsizeb = 4096
```

En el archivo xconf.h se declara devtab como un arreglo de estructuras del tipo devsw. A continuación se muestra la Tabla de dispositivos (la estructuta devsw) y devtab.

```
/* Declaraciones en la Tabla de dispositivos */
struct devsw  {          /* Entradas en la Tabla de
                            dispositivos */
   int  dvnum;           /* # del dispositivo */
   char dvname[DNMLEN];  /* nombre del dispositivo*/
   int  (*dvinit)();
   int  (*dvopen)();
   int  (*dvclose)();
   int  (*dvread)();
   int  (*dvwrite)();
   int  (*dvseek)();
   int  (*dvgetc)();
   int  (*dvputc)();
   int  (*dvcntl)();
   int  dvport;       /* dirección del 1er. puerto */
   int  dvivec;       /* vector de int. asociado */
   int  (*dviint)(); /* procesa la int. de entrada */
   int  (*dvoint)(); /* procesa la int. de salida */
   char *dvioblk;     /* bloque de control */
   int  dvid8259      /* id del disp. para el 8259 */
   int  dvminor;      /* cual disp. es, de un mismo
   };                    tipo */

extern struct devsw devtab[];   /* una entrada por
                                   dispositivo */
```

Existe una estructura devsw en devtab por cada dispositivo de E/S que se maneja en XINIX; los dispositivos son:

```
/* definiciones de nombres de dispositivos*/

#define CONSOLE     0        /* type tty */
#define OTHER_1     1        /* type tty */
#define OTHER_2     2        /* type tty */
#define DISK0       3        /* type dsk */
#define DISK1       4        /* type dsk */
#define DISK2       5        /* type hard disk */
#define PRINTER     6        /* type prt */
#define PRIM_ARCH   7        /* num. del primer
                                archivo en devtab */
```

El número de entrada en devtab es a su vez la identificación global de los dispositivos. En el archivo x2hlio.c estan las rutinas de E/S (read, write, seek, etc.) globales para todos los dispositivos, la idea es utilizar todas las funciones globales con todos los dispositivos y así evitar al usuario el detalle de cada uno de ellos. Devtab sirve entonces para ejecutar la rutina correspondiente de acuerdo al tipo de dispositivo.

Ahora suponer tres terminales: CONSOLE, OTHER1, OTHER2. la información de control y datos de cada dispositivo está en cada entrada del arreglo:

tty (para las terminales)
dstab (para los discos)

Qué entrada, en estos arreglos de información de control se va a usar ?. Eso nos lo dice dvminor un campo de la estructura devtab.

3.3.3.- Sus Funciones y Servicios.

Con XINIX original se efectúan las operaciones básicas de lectura/escritura a disco con bloques de 512 Bytes. Además cuenta con las operaciones también de lectura/escritura para tracks completos del diskette (de 9 sectores). Otro de los servicios que soporta XINIX, es el de formateo, el cual utiliza el SA de XINU. También se pueden verificar tracks completos o sectores, y como se mencionó en la sección anterior, se logra cambiar dinámicamente el tamaño del bloque de operación.

Referencias en Bibliografía:

[7] Kris Jamsa, 1988.
[8] Kumar-Barkakati, 1989.
[9] Norton-Wilton, 1988.

CAPITULO 4

MANEJADOR DE DISCO DURO PARA XINIX

4.1. Organización Física del Disco Duro.

4.1.1.- Componentes del Disco Duro.

La parte principal del disco duro es el conjunto de platos de aluminio (Ver Figura 4.1). Los platos están cubiertos con un material de óxido magnético que puede almacenar una carga negativa o positiva. Para maximizar la capacidad de almacenamiento del disco, ambos lados de los platos son usados para almacenar información. Los discos duros raramente tienen más de 32 platos, y típicamente tienen menos de 8 platos. La cara superior del plato superior es algunas veces llamado lado "cero"; y el otro lado del mismo plato superior es llamado lado "uno". A diferencia del disco flexible, los platos del del disco duro giran a una velocidad de 3600 r.p.m. (que es equivalente a 60 rev. por seg.) [10].

La transferencia de datos para y desde la superficie de los platos se hace a través del manejador de las cabezas del disco. El manejador de las cabezas está acoplado a una armadura que mueve las cabezas a través de la superficie de los platos (en lo sucesivo, "plato" lo llamaremos disco).

4.1.2.- Organización de la Superficie del Disco.

El manejador de disco duro (MDD) debe ser capaz de almacenar y recuperar datos rápidamente. La superficie del disco, por consiguiente, está organizada dentro de un sistema de coordenadas. Este sistema es usado para posicionar las cabezas sobre el área que contiene el dato deseado. El sistema de coordenadas divide cada plato en tracks, cilindros, y sectores.

Cada uno de los sectores del disco debe tener un número de indentificación. Este número es acompañado a través de un esquema de direccionamiento llamado "disk interleave" (intercalamiento de disco). El "interleave" tiene un impacto significativo sobre la eficiencia del MDD. Otros factores que afectan la eficiencia del disco, estan más directamente relacionados con las componentes del hardware.

4.1.3.- Pistas (o Tracks).

Cuando una cabeza se posiciona sobre el plato que está girando, la cabeza puede accesar un área de la superficie del disco que describe un círculo. Después de que el disco completó una revolución, las cabezas se reposicionan a sí mismas sobre otra parte del plato. Si la cabeza se mueve hacia el centro del disco, o hacia la orilla, esta puede leer otro "círculo" de datos sobre el disco. Un anillo completo de

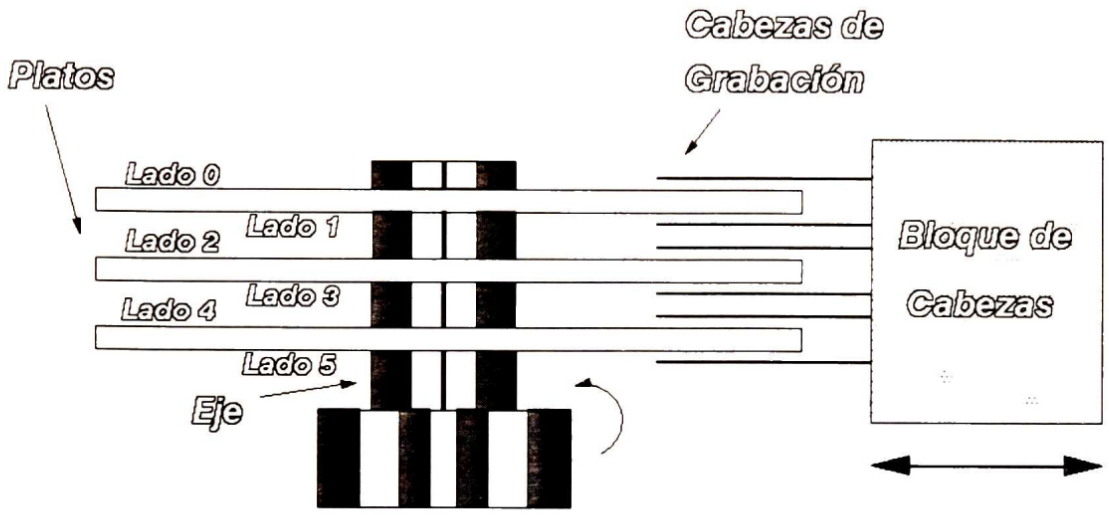


Figura 4.1.- Componentes de un Disco Duro Típico.

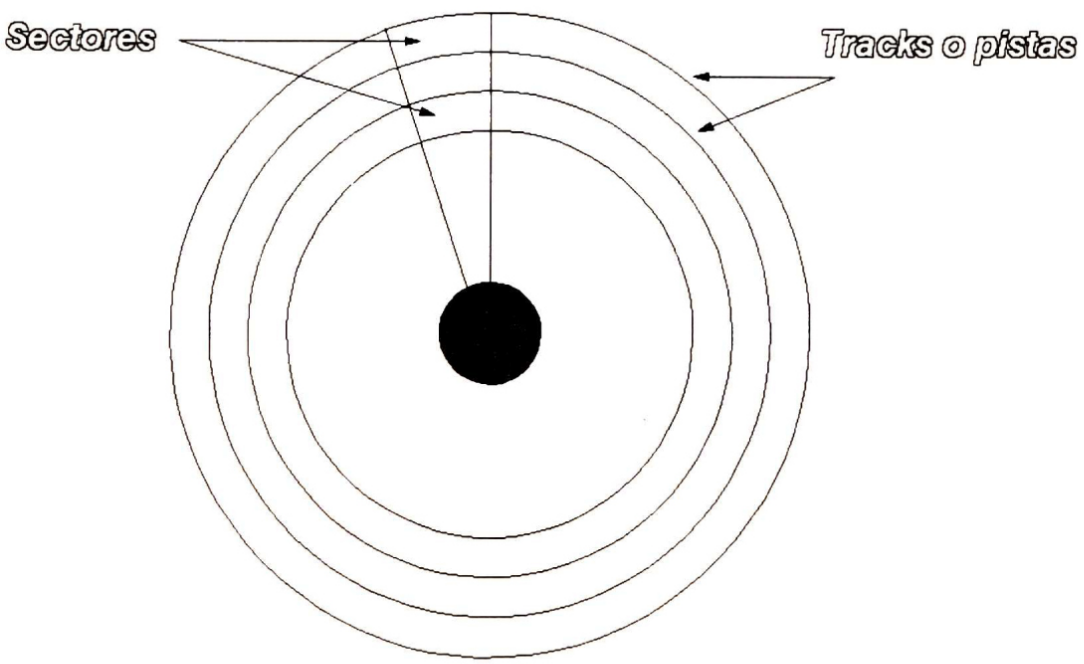


Figura 4.2.- Superficie de una cara del Disco Duro.

información sobre el disco es nombrado "track" (o pista).

La figura 4.2 ilustra la organización de los tracks sobre una superficie de un disco. Los tracks forman círculos concéntricos perfectos. El número de tracks disponible sobre una superficie de disco es determinado por las tolerancias del posicionador mecánico. Entre mayor sea el número de traks en el disco, obviamente, mayor será su capacidad de almacenamiento [10].

4.1.4.- Cilindros.

Todas las cabezas de los discos están montadas sobre un mecanismo sencillo (ver figura 4.1), el cual mueve las cabezas al mismo tiempo en una columna vertical. Cuando la cabeza del disco superior está sobre el track 5, las demás cabezas van a estar también en el track 5 de cada plato y en ambos lados de cada uno de ellos. De hecho, todas las cabezas están sobre el track 5, sobre todos los lados de los platos.

El retraso de tiempo principal en accesar datos sobre un disco duro, es el tiempo requerido para mover la cabeza a una posición sobre un track específico. Para optimizar la velocidad de lectura y escritura, los datos son frecuentemente accesados desde el mismo track (por ejemplo, el track 5) sobre ambos lados de cada plato, en lugar de utilizar múltiples tracks en un mismo plato. Esto significa que múltiples tracks pueden ser leídos sin mover las cabezas. Una columna vertical de tracks alineados es llamado cilindro.

Un cilindro es igual que un track tridimensional; el término "cilindro" y "track" se utilizan indistintamente. Si los platos de un disco tienen 2000 tracks, entonces los traks alineados verticalmente forman 2000 cilindros. Los cilindros usan el mismo esquema de numeración que los tracks, el cilindro más externo es el cilindro cero, el siguiente el cilindro 1, y así sucesivamente [10].

4.1.5.- Sectores.

El volúmen de datos que puede ser almacenado en un track sencillo, depende del esquema de codificación de los datos usado por el disco. Típicamente, un track de un disco duro puede almacenar alrededor de 9 KBytes (para un disco de 17 sectores). Para usar este espacio de almacenamiento disponible más eficientemente, la mínima unidad de almacenamiento de datos debe ser menor que un track. Los tracks, por consiguiente, están divididos en sectores. Un sector es un segmento de un track (Ver Figura 4.2). El número de sectores en un track depende del tipo de disco duro y controlador. Usualmente, un track contiene 17, 26, 31 o 34 sectores, y un sector contiene 512 bytes.

Aún y cuando los tracks más externos tienen mayor área, ellos almacenan la misma cantidad de datos que los tracks más internos, que obviamente tienen menor área de grabación. El SO "desperdicia" algo en capacidad de almacenamiento al mantener el mismo esquema de direccionamiento de los sectores

y el ancho del track, pero también simplifica la operación del disco e incrementa la eficiencia [10].

Para encontrar un dato específico, el disco debe conocer en que plato, track y sector buscará la información. Cada plato, track y sector es identificado por un número único.

4.1.6.- Formateo a Nivel Bajo (DEBUG).

Antes de usar un disco duro, éste debe estar formateado a nivel bajo. El formateo a nivel bajo consiste en: estructuración de la superficie, selección del "interleave", y mapeo de sectores defectuosos [10].

En la primera fase del formateo, el programa de formateo (DEBUG) organiza la superficie de cada plato en tracks y sectores. A este paso se le llama estructuración de la superficie. Durante el formateo a nivel bajo, cada sector recibe un número de identificación único. En la operación normal, el MDD usa estos números para encontrar sectores específicos en el disco. Esta organización de tracks y sectores es el primer esquema del mapeo del disco (o esquema de coordenadas) usado por el SO cuando se accesan los datos.

Debido a que el ID de cada sector depende de la razón del interleave del disco, el programa de formateo de bajo nivel siempre establece el "interleave" del disco (este término lo explicaremos en la siguiente sección). El interleave del disco indica el número de veces que los platos del disco deben girar para que se lea un track completo.

El proceso de comprobar los defectos de la superficie del disco es llamado mapeo de sectores defectuosos ("defect mapping"). Normalmente el fabricante del disco coloca una etiqueta en el disco, indicando una lista de sectores defectuosos, en el caso de que existan. Estos sectores defectuosos se introducen manualmente cuando se está llevando a cabo el formateo (DEBUG).

El último paso del formateo a nivel bajo consiste en verificar la integridad de la superficie del disco. Durante estas pruebas, el programa escribe los datos al disco, y después lee los mismos datos para asegurar que el dato fué grabado correctamente.

El comando DEBUG de MS-DOS efectúa el formateo a nivel bajo, en el Apéndice 1 se indica cómo hacerlo.

Por razones de eficiencia los sectores de los tracks no necesariamente están en orden secuencial. Por ejemplo, al sector 1 le puede seguir el sector 7 (En la siguiente sección se explica la razón de este procedimiento). Los primeros bytes de cada sector contienen el número de identificación de los sectores. Estos bytes iniciales (llamado encabezado del sector) eliminan la necesidad de un índice que mantenga la localización de los sectores de cada track (Ver Figura 4.3). Para encontrar un sector específico, el manejador mueve la cabeza a lo largo del track apropiado y lee cada encabezado de sector hasta que encuentra el sector correcto. Este esquema de codificación es llamado "soft sectoring".

La información en los "headers" es escrita en cada

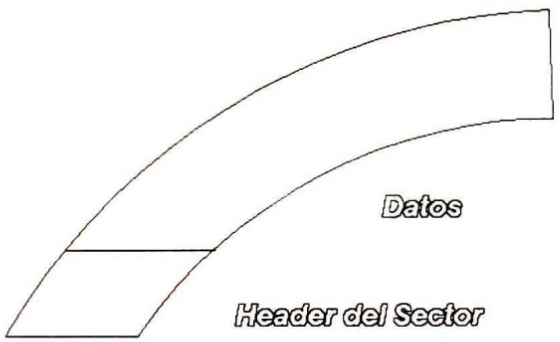


Figura 4.3.- El Encabezado del Sector tiene su número de identificación.

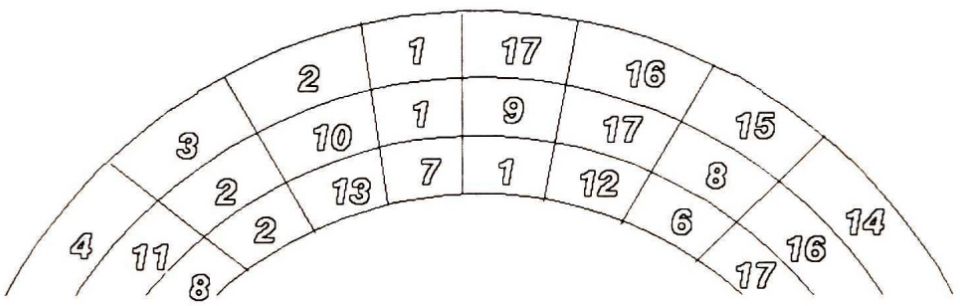


Figura 4.4.- Relaciones de "Interleave" sobre un Track de 17 Sectores.

sector durante el formateo de bajo nivel. La dirección del sector contiene el número de track, el número de cabeza y el número de sector. En el Capítulo 3, Figura 3.3, se describió esta información que contiene el "header" [10] [11].

4.1.6.1.- "Interleaving"

Cuando se guarda un archivo en el disco duro, el SO escribe el archivo en uno o más sectores. Supongamos que un archivo va a ser almacenado en tres sectores, y que todos los sectores en el track están libres. La cabeza se mueve al track indicado, y cuando la cabeza está sobre el sector 1, ésta escribe la primera parte del archivo a ese sector. El MDD entonces envía la siguiente parte del archivo a las cabezas. Sin embargo, por el tiempo que tardan los datos en llegar a las cabezas (recuérdese que el disco gira a 3600 r.p.m.), una pequeña pero considerable cantidad de tiempo es requerida antes de realizar la transferencia, ya que las cabezas pueden estar posicionadas a la mitad del camino sobre el sector 2. El disco debe hacer otra revolución completa antes de que el sector 2 esté otra vez bajo las cabezas. Entonces, en ese momento la cabeza puede escribir la segunda parte del archivo en el sector 2. Debido al tiempo que empleó el MDD en enviar la parte final del archivo, las cabezas, pueden estar un poco adelante del sector 3, y de esta manera, el sector tendrá que efectuar otra revolución completa antes de que la tercera parte del archivo se guarde en el disco. En el peor de los casos, el disco puede hacer hasta 17 rev. para guardar un track completo (En el caso de que se utilize un disco duro de 17 sectores).

Si la computadora, el controlador y el disco mismo operan a altos niveles de eficiencia, ellos pueden ser suficientemente rápidos para enviar los datos a la cabeza antes de que ellos pasen al inicio del siguiente sector. Muchos sistemas, sin embargo, simplemente no son rápidos. Con máquinas lentas, por consiguiente, los sectores no se ordenarán consecutivamente. Antes bien que numerar los sectores adyacentes uno tras otro, el SO puede intercalar ("interleave") los sectores así que el sector 2 esté localizado dos o más sectores después del sector 1. Cuando se usa el interleaving, el MDD tiene más tiempo para pasar los datos a los manejadores de las cabezas.

En la figura 4.4 se muestran tres diferentes relaciones de "interleave". Una mitad de disco tiene un interleave de 1:1 cuando sectores adyacentes están numerados consecutivamente (1,2,3,4 ... 17). Otra opción sería un interleave 2:1, esto es cuando los sectores están numerados alternadamente (1,10,2,11,3,12, y así sucesivamente).

Finalmente, podemos asegurar que la razón del interleave tiene un profundo efecto en la eficiencia del disco duro [10].

4.1.7.- Factores de Eficiencia.

El disco duro tiene dos factores que alteran notablemente su rendimiento. Estos son el tiempo de acceso promedio y el tiempo de transferencia. Vea la Figura 4.5 [10] [12].

4.1.7.1.- Tiempo de acceso promedio.

Cuando la PC hace una solicitud a disco, las cabezas deben moverse desde su posición actual al track indicado por dicha solicitud. El tiempo requerido en mover la cabeza a su nueva posición es llamado tiempo de búsqueda o tiempo de acceso.

El tiempo de acceso depende de la distancia desde la posición actual de las cabezas hasta el track objetivo, y puede variar con cada solicitud de transferencia de datos. Para cuantificar la eficiencia en una forma más fidedigna, el personal técnico ejecuta múltiples pruebas para encontrar el tiempo promedio que necesitan las cabezas para accesar un track aleatorio. El track objeto puede estar cerca o lejos de la posición actual de las cabezas. Las pruebas aleatorias normalmente calculan el tiempo requerido en tiempo de búsqueda sólo en una tercera parte de los tracks de un plato. El tiempo de acceso promedio es la duración promedio de estas pruebas de búsqueda aleatoria.

Una vez que las cabezas están en el track deseado, la computadora debe esperar a que el sector apropiado esté debajo (o arriba) de las cabezas. Este período de espera es llamado tiempo de latencia. Estadísticamente, el tiempo de espera deberá ser el tiempo que toma el disco en hacer la mitad de una revolución; esta medida de tiempo es llamada latencia promedio. Debido a que todos los discos duros giran a 3600 r.p.m., todos tienen la misma latencia promedio de 8 ms.. La latencia promedio, por consiguiente, no es factor útil estadísticamente para comparar la eficiencia del disco duro.

4.1.7.2.- Tiempo de transferencia.

La razón con la cual los datos pueden ser transferidos para y desde el disco duro es llamado razón de transferencia. La razón de transferencia es medida en megabits por segundo (mps).

Tal como se mencionó anteriormente, los discos duros de todas las PC's giran a la misma velocidad, el único factor que influye en la razón de transferencia es la cantidad de información que puede ser almacenada en cada sector o track. Esta capacidad de almacenamiento es la densidad del disco. La densidad es determinada principalmente por el tipo de interfaz del controlador y el método de codificación de datos usado.

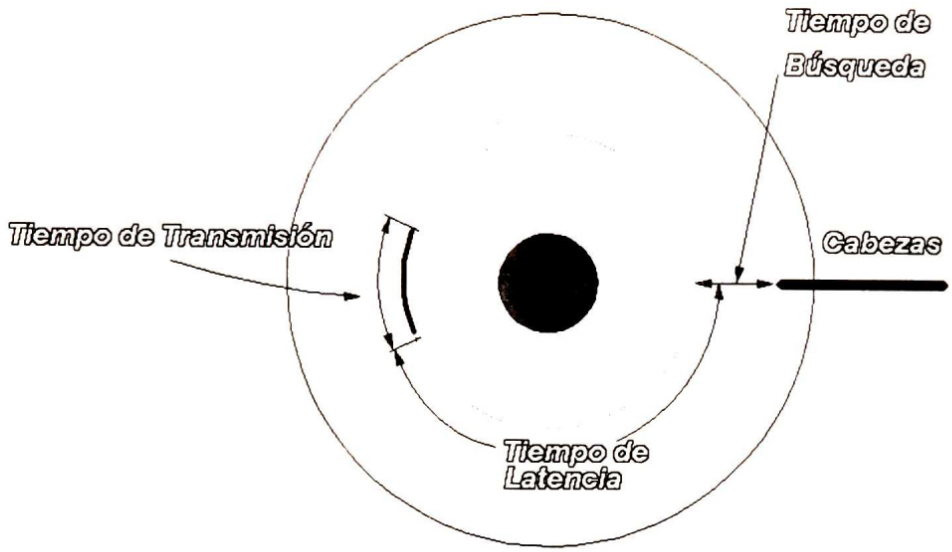


Figura 4.5.- Tiempos de Acceso del Disco Duro. [12]

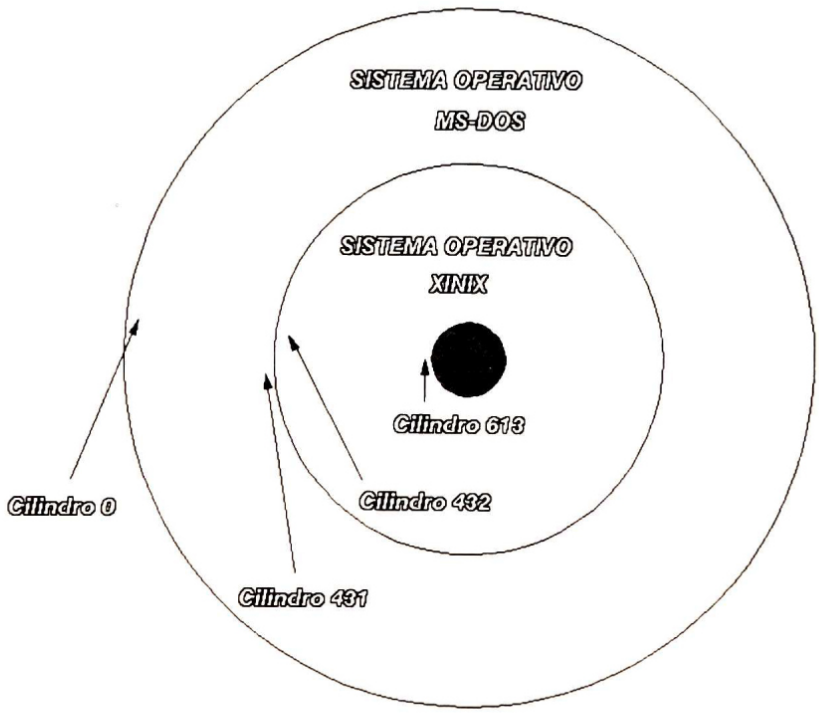


Figura 4.7.- Disco Duro particionado.

4.1.8.- Almacenamiento y recuperación de datos.

Mientras que los sectores y tracks son la parte fundamental de un sistema de almacenamiento de datos, ellos solos, no proveen suficiente control de un sistema de almacenamiento y recuperación de datos. El comando FORMAT de DOS se utiliza para agregar otra capa del esquema de direccionamiento (formateo lógico), el cual maneja los datos para que se organizen en archivos.

Este formateo lógico consiste de las mismas áreas de trabajo que ya se mencionaron para el caso de los discos flexibles, y que son:

- el registro "boot".
- la FAT.
- y el área de directorios.

La única diferencia entre el disco flexible y el disco duro, respecto al formateo lógico, es la tabla de particiones la cual se crea con el comando FDISK de DOS, y que se encuentra instalada en el primer sector fisico del disco (cabeza 0, cilindro 0, sector 1). En la Figura 4.6 se ilustran las tres áreas de trabajo para un disco duro de 17 sectores, 4 cabezas y 614 cilindros.

A continuación se volverá a describir cada área de trabajo, complementándose con la información ya expuesta en el Capítulo 3, sección, Qué es el Formateo Lógico.

4.1.8.1.- El Proceso de Formateo.

Cuando se formatea un disco a nivel bajo (o físico), el espacio físico de los discos es organizado en tracks, sectores por track, y bytes por sector. Estos son los elementos físicos que permiten la división y direccionamiento del espacio de almacenamiento del disco. El SO usa un esquema lógico para manejar archivos. Este esquema no solamente provee velocidad, también facilita a los programas accesar el disco sin hacer uso del esquema de direccionamiento físico. El programa FORMAT organiza el disco duro y establece este formateo lógico de la unidad. Ver el Apéndice 2 si el lector desea tener más información sobre el comando FORMAT de DOS.

4.1.8.2.- El sector "Boot"

El primer sector de un disco formateado bajo DOS, está definido como el registro "boot". Este contiene un programa corto que es cargado automáticamente en memoria cuando el disco se usa para cargar el SO MS-DOS después de que se enciende el sistema o se dá "reset" [8]. Este programa da instrucciones a la computadora de dónde buscar en el disco los archivos que conforman el SO MS-DOS. Una vez que localiza los archivos, el programa "boot" carga los archivos

en memoria y transfiere el control a MS-DOS. Debido a que el número de archivos de MS-DOS y la manera en la cual ellos están almacenados puede diferir de acuerdo al tipo de implementación (por ejemplo, IBM PC, COMPAQ, CompuPro, etc.), por tal motivo el contenido del registro "boot" puede variar. Con el propósito de ser consistentes, el sector "boot" está definido en el primer sector del disco formateado, a menos de que el disco formateado se utilize sólo para 'datos', es decir, no para un disco de "boot".

Los tres primeros bytes del registro contienen una instrucción "jump" (Vea la Figura 3.5). En el tiempo de inicialización, la instrucción "jump" le dice al sistema que brinque pasando la primera parte del registro del código de "boot". Con la introducción de la versión 2.0 de MS-DOS, los 27 bytes del registro "boot" entre la instrucción "jump" inicial y el código de "boot", contiene informacón acerca del formato del disco.

El registro "boot" es inicialmente creado cuando el disco es formateado usando el comando FORMAT para discos flexibles, o el comando FDISK para discos duros.

4.1.8.3.- La Tabla de Particiones.

A partir de la introducción de la versión 2.0 de MS-DOS, se agregó una nueva información al formateo del disco: la tabla de particiones del disco duro. La Tabla de particiones se usa para describir como se divide un disco duro en varias secciones, y casi siempre se utiliza en discos duros desde 10 Mbytes o actualmente con discos de mayor capacidad de almacenamiento.

Antes de continuar, nos podríamos preguntar ¿ Qué son las particiones y porqué hay necesidad de utilizarlas ?. Pues bien, las particiones son divisiones dentro de un disco duro a las cuales se pueden asignar diferentes SO's. Un SO puede usar solamente una partición. Las particiones permiten al usuario seleccionar el SO preferido. De hecho, para el desarrollo de esta tesis, se utilizó un disco con dos particiones ("C" y "D"), en la primera partición se instaló MS-DOS y en la seguna al SO XINIX. En la figura 4.7 se ilustra un disco duro dividido por los dos SO's mencionados, así también se indican los cilindros que delimitan a cada partición para este caso particular.

Otra de las ventajas que ofrecen las particiones es que hay más flexibilidad en seleccionar una amplia variedad de paquetes de "software" que corren bajo diferentes SO's.

El "Master Boot Record" se muestra en la Tabla 4.1, en la Tabla 4.2 se muestra el significado de los 16 bytes que contiene cada partición.

En la figura 4.8 se ilustra la tabla de particiones de un disco duro de 40 MBytes. Se utilizó el programa NU de las utilerías de NORTON (esta utilería en particular auxilia al usuario en la recuperación de archivos perdidos en disco, así como en efectuar accesos de escritura/lectura al disco; el

TABLA 4.1

El primer sector físico de un disco duro (Cilindro = 0, cabeza = 0, sector = 1) contiene el "Master Boot Record". En la dirección 0X1BE empieza la Tabla de Particiones, consiste de 16 Bytes por partición. A continuación se describe:

BYTES	C O N T E N I D O
000-1BDH	Reservado
1BE-1CDH	Partición # 1
1CE-1DDH	Partición # 2
1DE-1EDH	Partición # 3
1EE-1FDH	Partición # 4
1FE-1FFH	Palabra Firma (AA55H)

TABLA 4.2

BYTES	C O N T E N I D O
00H	Estado de la Partición (0=no "booteable", 80H=activa o "booteable".
01H	Cabeza Inicial.
02-03H	Cilindro/Sector inicial.
04H	Tipo de Partición: 00H No se usa. 01H Sistema de Achivo (SA) con FAT 12 bits 04H SA con FAT 16 bits 05H Partición extensión de MS-DOS 06H Partición "Huge" (Ver. 4.0 de MS-DOS)
05H	Cabeza final.
06-07H	Cilindro/Sector final.
08-0BH	Sector inicial de la partición, relativa al inicio del disco.
0C-0FH	Longitud de la Partición en sectores.

usuario proporciona el sector, cilindro y cabeza o el número de cluster deseado).

Cuando se particiona el disco duro en una o más particiones con el comando FDISK de DOS (Ver Apéndice 3, si se desea tener una mejor explicación del Comando FDISK), se tiene un nuevo registro de boot con la tabla de particiones

almacenado en el primer sector de cada partición. De esta manera, un disco con una partición contiene una tabla de particiones maestra y una segunda tabla de partición en su registro "boot" almacenado en el primer sector de la partición misma. Particiones adicionales también contienen su propio registro de "boot" y tabla de particiones. La tabla de particiones maestra es actualizada cada vez que se utiliza FDISK para cambiar las particiones, y el campo de "status" de la partición de cada partición individual es actualizado para indicar su estado de activa o inactiva.

4.1.8.4.- "Clusters".

La unidad más pequeña de almacenamiento del disco es un sector de 512 Bytes. El SO debe conocer cuales sectores están en uso (que sectores tienen información) y cuales sectores están disponibles para almacenar nueva información.

Un disco duro de 100 MBytes tiene alrededor de 200,000 sectores. El SO sería ineficiente si tuviera que manejar cada sector individualmente. Es cierto que una computadora puede manejar fácilmente una lista de 200,000 artículos, pero el acceso a los archivos pudiera ser lento. Para hacer al disco más manejable, el SO usa un esquema de referencia el cual considera un grupo de sectores juntos como una unidad sencilla. A tal grupo de sectores se le llama cluster.

En la sección 3.2.3, también se hace referencia al término de "cluster".

4.1.8.5.- FAT ("File Allocation Table").

La FAT mantiene un registro de todos los clusters del disco. Cuando se escribe un archivo al disco, el SO busca la FAT para determinar la localización del siguiente cluster libre. La FAT también le dice al SO qué cluster está ocupado por un archivo, e identifica los clusters reservados y los dañados [8].

Cada cluster en el área de datos del disco (que es el área que no está reservada para el registro de "boot" y FAT) recibe un número de referencia único. Este número es diferente de los números de sectores y tracks, los cuales son relativos a un lado específico o track del disco. La FAT mantiene una entrada (o registro) por cada cluster del disco, y el número de cluster es usado como el índice principal.

Cada cluster puede tener cualquiera de los valores que se indican en la Tabla 4.3.

TABLA 4.3.- VALORES QUE PUEDEN TENER LOS REGISTROS EN LA FAT. [8]

Valor en Hex. del registro del FAT.	Significado
(0)000	Cluster libre.
(F)FF0 al (F)FF6	Cluster reservado.
(F)FF7	Cluster marcado como dañado, no se utiliza.
(F)FF8 al (F)FFF	Ultimo cluster ocupado por un archivo, o lo que es equivalente, Fin de archivo.
(X)XXX	Cualquier otro valor indica un número de cluster en la cadena, definiendo cómo es almacenado un archivo.

4.1.8.6.- Directorios.

El SO usa ésta última área, llamada directorio, la cual tiene información de cada archivo almacenado en el disco. El directorio raíz del disco es automáticamente creado cuando se formatea el disco. Un directorio contiene una entrada (o registro) por cada archivo en el directorio; cada registro incluye la información mostrada en la Figura 3.7.

Cada nombre de directorio o archivo toma 32 bytes; existen 10 bytes que están reservados para uso futuro. Note que la información incluye el número de cluster inicial, el cual provee una "liga" o relación con la FAT.

El tamaño del directorio raíz se fija durante el formateo. Todos los discos, por consiguiente, tienen un número máximo de archivos que pueden ser creados en el directorio raíz. Y este número máximo varía de acuerdo al número se sectores asignados al área de directorios. Un número típico, es 512 archivos.

4.2.- Manejo del Disco Duro.

4.2.1.- Fórmulas de Conversión.

Antes de entrar en detalle respecto a las fórmulas que utiliza el disco duro para convertir desde un número dado de sector lógico a cilindro, cabeza y sector, es conveniente conocer cómo se lleva a cabo la inicialización del disco duro. Primero, en xinix.c, al inicializarse XINIX, uno de varios procesos que crea es el de disk_operator() (el cual está en x2dskdrv.c), este proceso lleva el control de las peticiones del disco flexible y del disco duro. Posteriormente, en este mismo capítulo se explicará cómo se efectúa

el manejo de las peticiones para ambos tipos de discos.

Después de ejecutarse el proceso disk_operator(), se verifica si existe disco duro en el sistema (se comprueba el contenido de la dirección absoluta de memoria 0x475 [9], la existencia del equipo periférico instalado, en particular el disco duro), y en el caso de que sea afirmativo, entonces se procede a la primera fase de la inicialización con la función xinit(DISK2). El principal propósito de xinit(DISK2), es verificar si realmente DISK2 corresponde a uno de los dispositivos de XINIX (DISK 2 es el nombre mnemónico que le corresponde al disco duro). La segunda fase de la inicialización la lleva a cabo init_hd_params() (ésta función se encuentra en x2win.c, y la llama disk_operator()), esa función inicializa la estructura **part_entry**, declarada como global en x2win.c. La estructura part_entry tiene toda la información sobre las particiones del disco duro; a continuación se muestra la estructura:

```
struct part_entry  {
  char bootind; /* indicador de boot 0/0x80 inac./activa*
  char start_head; /* valor de cabeza para el 1o. sector*
  char start_sec; /* valor del sector inicial */
  char start_cyl; /* valor del cil. para el 1o. sector */
  char sysind; /* indicador  del sistema de FAT usado;
                  1 = FAT 12 bits, 4 = FAT de 16 bits */
  char last_head; /* valor de cabeza para el último sec.*
  char last_sec; /* valor del último sector */
  char last_cyl; /* valor del cil. para el último sector*
  long lowsec; /* primer sector lógico */
  long size; /* tamaño de la partición en sectores */
};
```

Para obtener la información de la estructura part_entry, init_hd_params() primero procede a efectuar una operación de lectura a través de la función hd_bios_op(), tal como se indica:

```
hd_bios_op(DISK_READ, 1 /*Num.Sec.*/, nr_drive, 0 /*cil*/,
                1 /*Sec.*/, 0 /*Cabeza*/, secbuf );
```

analizando los parámetros de hd_bios_op(), notamos que además de efectuarse una operación de lectura y leer únicamente un sector, se tiene que se va a leer en el cilindro 0, sector 1, y en la cabeza o lado 0. Estas coordenadas corresponden al lugar físico donde se encuentra la tabla de particiones de los discos duros (Obsérvese la figura 4.6). El sector así leído se guarda en secbuf [16].

Dado que la dirección de la primera partición empieza en

0x1BE (Vea la Tabla 4.1) continuamos decodificando los 16 bytes siguientes y de esta forma obtenemos la información de la primera partición. Solo que XINIX se instala en la segunda partición del disco duro, por tal motivo la información que necesitamos de la primera partición es: el número de cabezas, el tipo de FAT, y el número de sectores del disco duro; estos tres parámetros se utilizarán después.

La segunda partición empieza en 0x1CE y de aquí se obtiene el cilindro inicial, el tamaño de la segunda partición y el primer sector lógico desde donde empezará XINIX. Con la obtención de esta información termina la función init_hd_params().

Ahora pasamos a w_do_rdwt(), primero se obtiene la siguiente petición a disco y el número de dispositivo que hace la solicitud.

Antes de continuar, cabe mencionar que el NSAX cuenta con el comando **cdxf_hd**, este comando copia un archivo desde la primera partición (la que corresponde a DOS) a la segunda partición de XINIX. Y para hacer esto primero se verifica si se esta utilizando el comando mencionado, si es así, entonces se obtiene el sector lógico tal como es, desde la petición; significa que vamos a invadir la partición de DOS para buscar el archivo que se copiará a la partición de XINIX.

Por otra parte, si la petición a disco duro proviene de un servicio de XINIX, entonces se obtiene el sector lógico de la petición y a este valor se le agrega el que corresponde al primer sector lógico desde donde empieza la partición de XINIX (sec_fisico_inicial_part_xin). Así aseguramos que cualquier petición al disco duro de XINIX quede dentro de los límites de la segunda partición. Esta variable juega un papel importante respecto a la integridad de la información del disco duro.

Una vez que se obtiene el sector lógico de la petición se procede a sacar las coordenadas de cilindro, sector, cabeza y se lleva a cabo tal como se muestra a continuación:

```
bigcyl = sector/(Nhead*Num_Sec); /* sector = sec. logico*

s = (sector%Num_Sec) + 1; /* s = sector */
c = bigcyl & 0xFFF;       /* c = cilindro */
h = (sector %(Nhead*Num_Sec))/Num_Sec  /* h = head */
```

Finalmente, si se utiliza el comando cdxf_hd, entonces la operaciones de lectura/escritura serán de un sector de 512 Bytes. Y por otro lado, si es una petición normal de XINIX entonces la operaciones de lectura/escritura serán de bloques de 1024 bytes.

4.3.- Estructura del Manejador de Disco Duro de XINIX.

4.3.1.- Peticiones Siguen Igual, se Usa el BIOS.

El manejo de las peticiones a disco flexible y disco duro se implementó en x2dskdrv.c en el proceso disk_operator(). Dicho proceso encola las peticiones de disco duro y flexible, indistintamente en la estructura "dreq" la cual se muestra a continuación:

```
struct dreq   { /* Nodo en la lista de solicitudes */

   int drfnum; /* # de disco flex.: 0 o 1; 2 para duro */
   unsigned int drdba; /* direc. bloque del disco a usar*/
   int drpid; /* ident. del proceso que hace la soli. */
   int drsizet; /*Tamaño de la transferencia (1) 512,
                  (2) 1024, (3) 2048, (4) 4096. */
   char *drbuff; /*direc. del buffer para lect/escritura*/
   char drop; /* operation: READ/WRITE/SEEK */
   int drstat; /* status q' regresa OK/SYSERR */
   int drstdio; /*disp. a enviar mensaje u obtener opcio*/
   struct dreq *drnext; /* apunt. al sig. nodo en la lista
                           de peticiones */
};
```

en drfnum identificamos si la petición es para un disco flexible o duro.

Teniendo como marco de referencia a la estructura anterior podemos dar una breve explicación de disk_operator(). El primer paso consiste en la inicialización del disco duro (init_hd_params()) en el caso de que esté instalado en el sistema, enseguida se entra en un ciclo (while(1)) el cual espera cualquier petición de ambos tipos de disco, si no hay petición entonces el proceso disk_operator() se suspende. Cuando llega una petición a disco duro, se llama a w_do_rdwt (en x2win.c) y si es de disco flexible, entonces a dskstrt().

Supongamos que la petición es para el disco duro, en w_do_rdwt() se llama a la función hd_bios_op(), esta utiliza los servicios del BIOS por medio de la interrupción 0x13. De los servicios que ofrece la Interrupción 13H solamente se utilizan los servicios de lectura/escritura [16].

4.3.2.- Porqué se Usa el BIOS.

Uno de los secretos de una buena programación, para la familia de las PC's cae en el uso efectivo del "software" que es construído dentro de la máquina: los servicios del ROM-BIOS [9]. Conceptualmente, los servicios del ROM-BIOS están entre el hardware y los lenguajes de alto nivel (incluyendo al SO). Vea la figura 4.10.

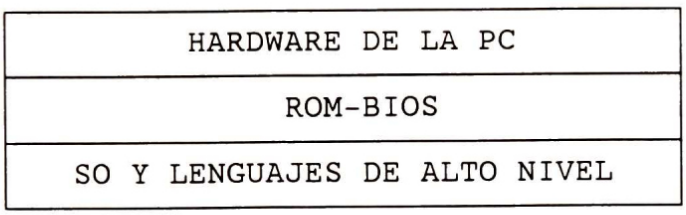


FIGURA 4.10

Estos servicios trabajan directamente con el hardware de la computadora y los dispositivos periféricos, ejecutando algunas de las tareas más fundamentales del sistema, tal como leer y escribir bytes individuales de datos al monitor o disco. Los servicios de DOS y los servicios de lenguajes de programación son frecuentemente implementados desde estas funciones básicas del BIOS, ya sea directamente o indirectamente, el usuario estará seguro de cualquier problema de compatibilidad.

Esto no significa que siempre utilizaremos los servicios del BIOS cuando estén disponibles. Las funciones de entrada/salida que provee DOS y los lenguajes de programación de alto nivel frecuentemente proveen los mismos servicios que ofrece el ROM-BIOS, pero en una forma que es fácil de usar en sus programas. Sin embargo, cuando un programa necesita un acceso directo a los dispositivos de entrada/salida de la PC (por ejemplo, el disco duro) que DOS o el lenguaje de programación puede proveer; los servicios del BIOS son usualmente una mejor alternativa.

Para finalizar podemos decir que se utilizó al Int. 13H del ROM-BIOS (Servicios del disco) para hacer transportable el software del manejador del disco duro en cualquier PC compatible con IBM y además de tener un acceso más directo con el disco duro.

4.4.- Sistema de Archivo Original de XINIX con Disco Duro.

4.4.1.- Implicaciones.

En el SA original de XINIX (SAOX) el tamaño del bloque lógico es igual al físico, el cual es de 512 bytes. Por otra parte, sólo se puede conservar un bloque en memoria por cada archivo que se esté accesando. De acuerdo al tamaño del archivo se tendrán (tamaño/512 = n) n accesos a disco para poder leerlo completamente en forma secuencial. No obstante, manejando bloques de 512 bytes tenemos menos fragmentación en disco, comparado con un sistema que utilize bloques de 1024 bytes, este sería el único punto a favor del SAOX. Ahora bien, con bloques de 512 bytes el número de accesos a disco duro será obviamente mayor que el manejar bloques de 1024.

El SAOX enlaza todos los bloques disponibles, en un disco flexible, creando una gran lista ligada de bloques, los cuales conforme se crean archivos se van asignando a ellos, eliminándose de la lista de disponibles, y asignándose a la lista de bloques que el archivo ocupa (i-blocks). Cuando se elimina parte de información de un archivo, o completamente, los bloques que pertenecían a ese archivo se liberan (se desenlazan de la lista del archivo) y se reincorporan a la lista de bloques disponibles. De acuerdo a lo expuesto, la asignación y desasignación de bloques se lleva a cabo manejando listas ligadas en ambos casos, la desventaja es que la lectura y/o escritura de archivos se vuelve bastante lenta por el manejo de los apuntadores lo cual no es conveniente para un SA instalado en disco duro.

Otra desventaja de SAOX es que éste tiene un directorio único con 28 entradas para archivos (en disco flexible), en donde cada entrada define las características del archivo apropiado, tiene además los apuntadores correspondientes a la lista de bloques de datos e i-blocks libres. Como consecuencia de la incapacidad de poder crear subdirectorios, no es práctico implementar el SAOX en el disco duro, ya que de hacerse estaría subutilizado o limitándose en gran medida la capacidad de almacenamiento del disco, además de no tener una organización respecto al manejo de archivos, debido a que todos los archivos existentes estarían en un mismo directorio, creando conflicto entre los usuarios.

Referencias en Bibliografía:

[10] Ainsbury, 1990.
[11] Gookin & Townsend, 1987.
[12] Deitel, 1987.
[8] Kumar, Barkakati, 1989.
[7] Jamsa, 1988.
[16] Duncan, 1988.

CAPITULO 5

IMPLEMENTACION DEL SISTEMA DE ARCHIVOS TIPO UNIX DE XINIX EN DISCO DURO

La implementación del Nuevo Sistema de Archivos de Xinix (NSAX) en el disco duro, implica el cómo se maneja el espacio en disco, cómo se almacenan los archivos y qué hacer para que todo funcione eficiente y confiablemente. En las siguientes secciones examinaremos varias áreas importantes relacionadas con esta implementación. La primera parte vista desde el enfoque del SO UNIX, la segunda parte veremos los cambios que el SO MINIX hizo a UNIX. Finalmente, en la última parte se verá cómo se lleva a cabo el formateo lógico en la segunda partición del disco duro, quedando así instalado el NSAX en el disco duro.

5.1.- Implementación de un Sistema de Archivo en General.

5.1.1.- Manejo del Espacio del Disco.

Existen dos maneras generales de almacenar un archivo de **n** bytes: se distribuyen **n** bytes consecutivos del espacio del disco o bien el archivo se divide en varios bloques, no necesariamente contiguos.

Una ventaja de la asignación contigua es que los registros sucesivos lógicos son, por lo general, físicamente adyacentes entre sí. Esto acelera el acceso, en comparación con los sistemas en los que los registros sucesivos lógicos están dispersos por todo el disco.

Los directorios de archivos de sistemas de asignación contigua son de implementación relativamente sencilla. Para cada archivo basta con conocer la dirección inicial del archivo y su longitud.

Por otra parte, la asignación contigua tiene otras desventajas. Al eliminar archivos, el espacio que ocupaban en el disco, es reclamado. Este espacio queda disponible para la asignación de nuevos archivos, pero estos nuevos archivos deben caber en los espacios disponibles. Así pues, los esquemas de asignación contigua muestran problemas de fragmentación, que es muy común en este sistema de asignamiento. Por esta razón, casi todos los sistemas de archivo recortan los archivos en bloques de tamaño fijo que no necesariamente deben ser adyacentes.

Una vez que se ha decidido almacenar archivos en bloques de tamaño fijo, surge la pregunta ¿ De qué tamaño debe ser el bloque ?.

Si disponemos de una unidad de asignación grande, digamos el tamaño de un cilindro (32KB) del disco, significa que a todos los archivos, aún menores a una decena de bytes, se asigna un cilindro completo. Regresando a la realidad, y de acuerdo a los estudios realizados por Mullender y

Tanenbaum [6] el tamaño de un archivo mediano en ambientes de UNIX es cerca de 1 KB.

Por otro lado, el uso de una unidad de asignación pequeña quiere decir que cada archivo constará de muchos bloques. La lectura de cada bloque normalmente requiere una localizaión y una demora rotatoria (Ver figura 4.5), de modo que la lectura de un archivo que consiste de muchos bloques pequeños será lenta. Ahora bien, utilizando unidades pequeñas de bloques tendríamos la ventaja de disminuir notablemente la fragmentación del disco y a la vez tendríamos que las tasas de transmisión de datos serían bajas, y viceversa. La eficienia del tiempo y la del espacio están fuertemente ligadas.

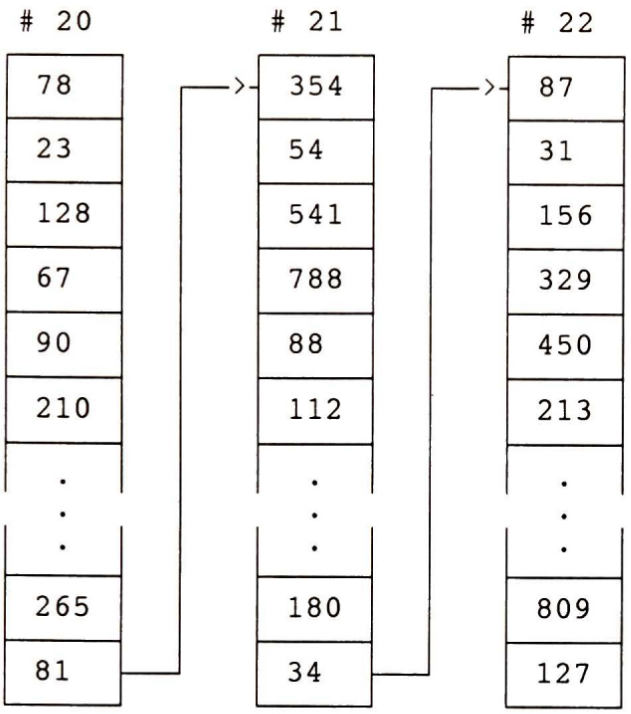
El compromiso usual consiste en seleccionar un tamaño de bloque de 512, 1K, 2K, 4K u 8K. Si se elige un tamaño de 1K (este tamaño de bloque lógico se utilizó para el NSAX) en un disco con un tamaño de sector de 512 Bytes, el SA siempre leerá o escribirá dos sectores consecutivos y los considerará como una unidad indivisible. En MS-DOS a ese tamaño de bloque lógico se le llama "cluster".

Ya elegido el tamaño del bloque, el siguiente paso consiste en cómo llevar el control de los bloques libres. Veremos dos métodos. El primero consiste en utilizar una lista enlazada de bloques de disco, observe la figura 5.1(a), donde cada bloque contenga tantos números de bloques libres como quepan. Con un bloque de 1K y un número de bloque de disco de 16 bits, cada bloque de la lista de bloques libres contiene los números de 512 bloques disponibles. Un ejemplo, para un disco de 20MB necesitamos una lista libre de 40 bloques para contener los 20,000 números de bloque del disco.

La otra técnica para el manejo del espacio libre es el mapa de bits. En la fig. 5.1.b se ilustra un mapa de bits. Un disco con **n** bloques requiere un mapa de bits con **n** bits. Los bloques libres se representan por medio de ceros en el mapa; los bloques asignados, por unos, o viceversa. Un disco con 20 MB requiere 20,000 bits para el mapa, que sólo ocuparía tres bloques. No es sorprendente que el mapa de bits ocupe menos espacio, ya que utiliza un bit por bloque en comparación con 16 bits en el modelo de la lista enlazada.

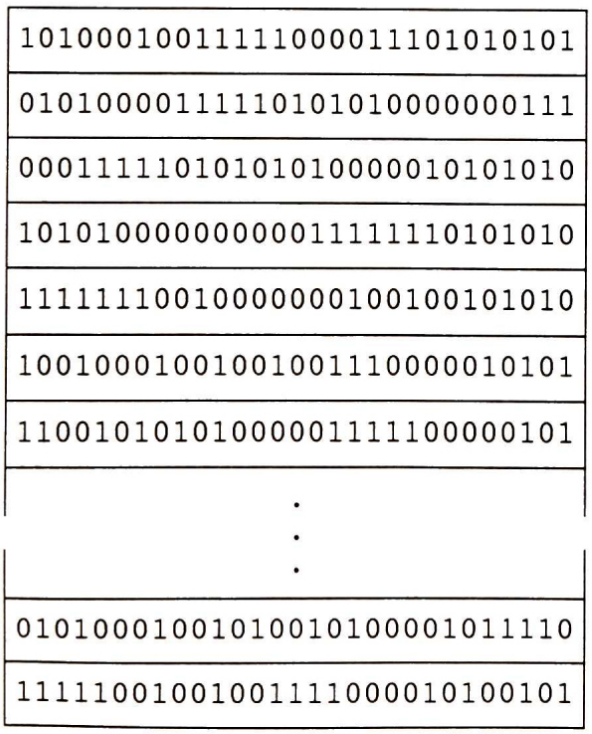
Si existe suficiente memoria RAM para contener el mapa de bits, ese método es preferible. Sin embargo, si sólo se puede reservar un bloque de memoria para llevar el control de los bloques libres del disco y el disco está casi lleno, entonces la lista enlazada puede ser más adecuada. Con sólo un bloque del mapa de bits en la memoria, puede resultar que no se puedan hallar bloques libres en él, lo cual produce un acceso del disco para leer el resto del mapa de bits. Cuando un bloque nuevo de la lista enlazada se carga en la memoria, pueden asignarse 512 bloques de disco antes de tener que ir al disco para capturar el siguiente bloque de la lista.

Para el NSAX se eligió el método del mapa de bits, el cual llevará el control de los bloques libres.



Un bloque del disco de 1KB contiene 512 enteros, o sea, 512 números de 16 bits.

FIGURA 5.1 (a).- LISTA ENLAZADA, MOSTRANDO LOS BLOQUES DISPONIBLES EN EL DISCO.



Ejemplo de un mapa - de bits, representado en un bloque de 1KB.

FIGURA 5.1 (b)

5.1.2.- Almacenamiento en Archivos.

Si un archivo consiste de una sucesión de bloques, el sistema de archivo debe contar con alguna manera de llevar el control de los bloques de cada archivo. La forma más lógica (el almacenamiento consecutivo de los bloques) suele no ser viable porque los archivos pueden crecer. De hecho, este fué el problema que nos llevó a dividir los archivos en bloques.

Un método que resulta adecuado consiste en almacenar los bloques de un archivo como una lista enlazada. Cada bloque del disco de 1024 bytes contiene 1022 bytes de datos y un apuntador de 2 bytes al siguiente bloque de la cadena. Sin embargo, este método tiene dos desventajas. Primero, el número de bytes de datos en un bloque ya no es una potencia de dos, lo que es impráctico. Y segundo y más grave, el acceso al azar es costoso de implementar. Por ejemplo, si un programa hace la localización del byte 32,768 y después inicia la lectura, el SO tiene que hallar su camino a través de 32768/ 1022 o 33 bloques para hallar los datos que necesita. Tener que leer 33 bloques del disco para hacer la localización es ineficiente.

Sin embargo, la idea de representar un archivo como una lista enlazada puede salvarse si se conservan los apuntadores en la memoria. La figura 3.6 muestra el esquema de asignación que utiliza MS-DOS. En ese ejemplo, se tienen dos archivos, el "file1" ocupa los clusters 2,3,4,8,9 y 10; mientras que el "file2" empieza en el cluster 5 y continúa con el 6 y 7.

Bajo el SO MS-DOS cada disco lleva asociada una Tabla de asignación de archivos (FAT). Dicha tabla tiene una entrada por cada bloque o cluster del disco. La entrada del directorio de cada archivo da el número de bloque del primero del archivo. Esa ranura en la FAT contiene el número del bloque del siguiente bloque.

Este esquema se diseñó originalmente para discos flexibles de 320 KB que utilizan un tamaño de bloque (o cluster) de 1 KB, el cuál es estándar para el S.O. MS-DOS. Los números de los bloques son de 12 bits, de modo que un FAT de 320 entradas requiere 480 bytes, y cabe en un sector de 512 bytes. Cuando IBM decidió formatear los discos flexibles a 360 KB, comenzando con la ver. 2.0 de MS-DOS, la FAT creció a 540 B, que ya no cabía en un sector (de 512 Bytes) y necesitaba cambiar el esquema del disco para dar cabida a una FAT mayor (de 2 sectores). Cuando se presentaron los discos duros con más de 4096 bloques hasta 20 MB, el número del bloque de 12 bits se volvió inadecuado y la FAT tuvo que alterarse de nuevo. Con la introducción de los discos de 40 MB se cambió la FAT a 16 bits. Ahora con discos de capacidades mucho mayores es necesario modificar la FAT arriba de 16 bits. Veamos el ejemplo del disco de 40 MB, este disco utiliza 64 sectores de 512 bytes para almacenar la primera copia de la FAT y otros 64 sectores más para la segunda copia de la FAT, lo cual da un total de 64 KB destinados exclusivamente para guardar la FAT. La

conservación de todo esto en la memoria todo el tiempo utiliza una buena parte de la memoria. Y si se conserva en el disco significa que la realización de una localización a la posición 32K de un archivo podría requerir un acceso al disco o tantas como 33 lecturas del disco para seguir la cadena de la FAT.

En resumen, el problema con la FAT es que los apuntadores de todos los archivos del disco en su totalidad se combinan al azar en la misma tabla. Esto infiere que toda la FAT se necesita forzosamente, aún si sólo se abre un archivo.

El lector interesado sobre operaciones efectuadas en la FAT, puede hacer referencia a la sección 5.5.2, en ella se explica el comando "cdxf_hd", con el cual se puede copiar un archivo de la partición de MS-DOS (la primera del disco duro) a la segunda partición (la que pertenece a XINIX) y para hacer la localización del directorio primero y después del archivo, en la primera partición, se emplea la FAT.

Ahora bien, un método más eficiente que la FAT sería conservar las listas de bloques de diferentes archivos en lugares distintos. Esto es lo que utliliza UNIX.

En asociación con cada archivo de UNIX hay una tabla pequeña (en el disco) llamada nodo-i, que se muestra en la figura 5.2. Esta contiene información sobre el tamaño del archivo, fecha de creación, etc., más adelante veremos el contenido con detalle del nodo-i. Por el momento, los elementos importantes son los 10 números de bloques del disco y los 3 números de bloque indirectos. Para archivos de más de 10 bloques de longitud, todas las direcciones del disco se conservan justamente en el nodo-i, siendo sencilla su localización.

Cuando un archivo crece hasta más de 10 bloques de disco se adquiere un bloque de disco libre y se coloca el apuntador indirecto hacia él. Este bloque se utiliza para guardar apuntadores de los bloques del disco. Con un bloque de disco de 1K y direcciones del disco de 32 bits, el bloque indirecto puede contener 256 direcciones del disco. Este esquema basta para archivos de hasta 266 bloques (10 en el nodo-i mismo, 256 en el bloque indirecto individual).

Después de 256 bloques, el apuntador doble indirecto se utiliza para apuntar a un bloque de disco de hasta 256 apuntadores. Sólo que estos apuntadores no apuntan a bloques de datos. Estos apuntan a 256 bloques indirectos individuales. El bloque indirecto doble es para archivos hasta de 266 + 256 (al cuadrado) que es igual a 65,802 bloques. Para archivos de más de 64 MB, se utiliza el apuntador triple indirecto para apuntar a un bloque que contiene apuntadores a 256 bloques indirectos dobles.

Los archivos que sobrepasen los 16 Gigabytes no se pueden manejar. Por supuesto, con un bloque de disco de 2K, cada bloque apuntador contiene 512 apuntadores en vez de 256 y el tamaño máximo se convierte en 128 Gigabytes. Imagínese Ud. una FAT de disco de 128 Gigabytes, la cantidad de bloques que utilizaría.

ESTRUCTURA DE UN NODO-I DE UNIX.

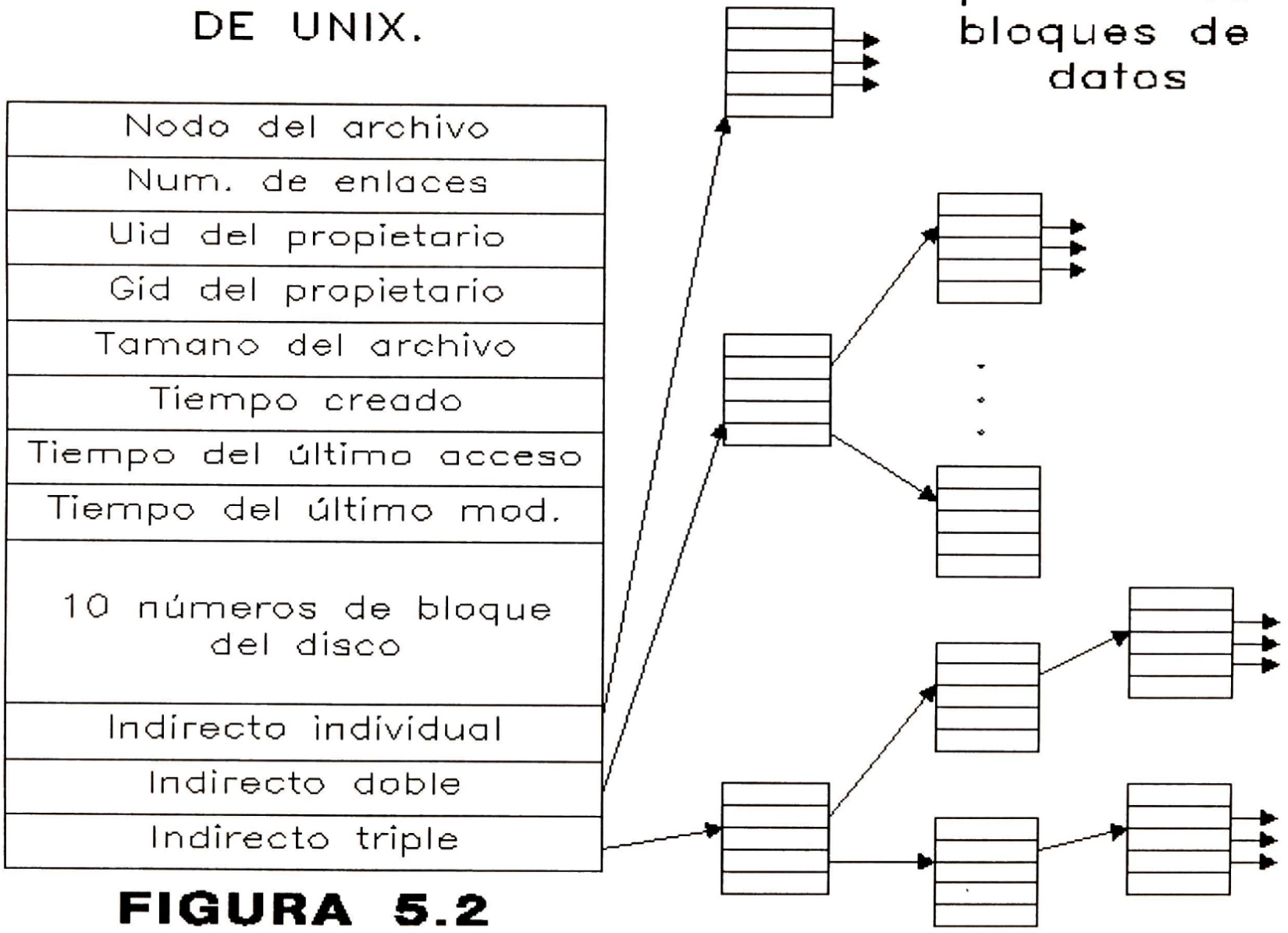


FIGURA 5.2

El poder del esquema de UNIX es que los bloques indirectos se utilizan sólo cuando se necesitan. Para archivos menores de 10 K, no se necesitan bloques indirectos en absoluto. Obsérvese también que para archivos aún más grandes, cuando mucho se necesitan tres referencias del disco para localizar la dirección de cualquier byte del archivo (sin incluir el acceso para obtener el nodo-i, el cual se captura cuando se abre el archivo y se conserva en la memoria hasta que se cierra).

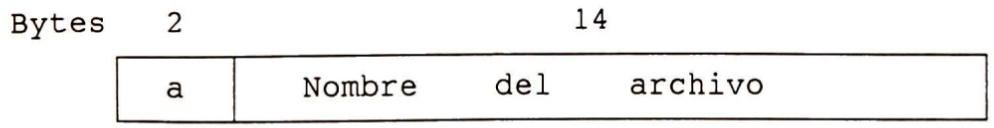
El esquema de almacenamiento del SO MINIX y XINIX es el mismo que el de UNIX, salvo que sólo se utilizan 7 bloques del disco en el nodo-i y no hay bloque indirecto triple. Con direcciones de disco de 2 bytes y bloques de 1 K, pueden manejarse archivos hasta de 262 MB, lo cual es más que suficiente para aplicaciones en computadoras personales.

5.1.3.- Estructura del Directorio.

Antes de hacer una operación de lectura a un archivo, éste primero se debe abrir. Cuando se abre un archivo, el SO utiliza el nombre de ruta ("path") proporcionado por el usuario para localizar los bloques del disco, de manera que pueda leer y escribir el archivo posteriormente. El delinear los nombres de ruta en nodos-i nos lleva al tema de la forma en que se organizan los sistemas de directorio.

Comencemos con el sistema de directorios del MS-DOS. La figura 3.7 muestra una entrada de directorio, la cual tiene 32 bytes de longitud, además, el nombre del archivo y el primer número de bloque, entre otros elementos. El primer número de bloque se puede emplear como índice en la FAT, para hallar el segundo número de bloque y así sucesivamente. De esta manera, se pueden encontrar todos los bloques de un archivo dado. Salvo por el archivo raíz, que es de tamaño fijo (112 entradas para un disco de 360 KB), los directorios de MS-DOS son archivos y pueden contener un número arbitrario de entradas, únicamente limitados por la capacidad del diskette o disco duro.

La estructura del directorio que se usa en UNIX y MINIX es extremadamente simple, en la figura 5.3 se ilustra la estructura del directorio de UNIX. Toda la información sobre el tipo, tamaño, tiempos, propietarios y bloques del disco está contenida en el nodo-i (Vea la figura 5.2). Todos los directorios de UNIX son archivos y pueden contener, en forma arbitraria, muchas de estas entradas de 16 bytes.



a).- Número del nodo-i

FIGURA 5.3.- ENTRADA DEL DIRECTORIO DE UNIX. ESTA MISMA ESTRUCTURA SE UTILIZA EN XINIX.

Al abrir un archivo, el sistema de archivo debe tomar el nombre proporcionado y localizar sus bloques del disco. Tomemos el ejemplo del siguiente nombre de ruta:

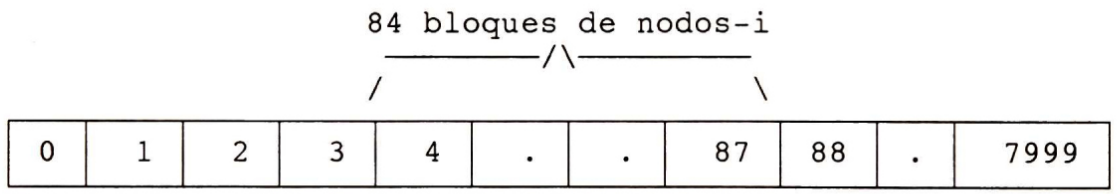
/usr_c/dir_1/xinix.c

Utilizaremos a UNIX como ejemplo, aunque el algoritmo es básicamente el mismo para todos los sistemas de directorios jerárquicos. En UNIX su nodo-i está localizado en un lugar fijo del disco.

La primer componente de la ruta , usr_c, se busca en el directorio raíz con el fin de hallar el nodo-i del archivo /usr_c. A partir de este nodo-i, el sistema localiza la siguiente componente, dir_1, en él. Cuando encuentra la entrada de dir_1, éste tiene el nodo-i del directorio /usr_c/dir_1. A partir de este nodo-i puede hallar el directorio mismo y buscar xinix.c. El nodo-i de ese archivo se lee después en la memoria y se guarda ahí hasta que el archivo se cierra.

5.2.- Estructura Lógica del Sistema de Archivo.

El sistema de archivos de XINIX es una entidad lógica compuesta por nodos-i, directorios y bloques de datos. Se puede almacenar en cualquier dispositivo, disco flexible o disco duro. En ambos casos, el esquema del sistema de archivo tiene la misma estructura. La figura 5.4 exhibe éste esquema para un disco duro que tiene 8 MBytes disponibles en la segunda partición. Para 8 MB el número de nodos-i es de 2674 (num_nodos_i = num_bloques/3 + 8), y se utiliza el tamaño del bloque de 1024 Bytes.



Bloque 0 = Bloque de "boot" (reservado para uso futuro).
Bloque 1 = Superbloque.
Bloque 2 = Bloque del Mapa de bits de nodos-i.
Bloque 3 = Bloque del Mapa de bits de los bloques de datos
Bloque 4 = Primer bloque de nodos-i.
Bloque 87 = Ultimo bloque de nodos-i.
Bloque 88 = Bloque directorio o primer bloque de datos del usuario.

Total de bloques	=	8000
Bloques usados por el NSAX	=	89
Bloques disponibles	=	7911

FIGURA 5.4.- ESQUEMA DEL NSAX, EJEMPLIFICADO PARA 8000 BLOQUES DE 1024 BYTES.

5.2.1.- El Bloque de Carga.

Cada sistema de archivo comienza con un bloque de carga ("boot"). En nuestro caso el Bloque "boot" se reserva para una aplicación futura, dónde se desee que el SO XINIX se inicialice desde el encendido de la computadora, posteriormente lea el bloque de carga y lo deposite en la memoria y salte hacia él. Una vez que el sistema se ha cargado, el bloque de carga ya no se utiliza más.

Cabe mencionar que los sistema de archivo mayores al del ejemplo o bien aquellos con más o menos nodos-i o un tamaño de bloque diferente, serán las 6 mismas componentes en el mismo orden, sólo que sus tamaños relativos pueden ser distintos.

5.2.2.- El Superbloque.

El superbloque contiene información que describe la distribución del SA. Este se ilustra en la figura 5.5.

La función principal del superbloque consiste en indicar al sistema de archivo qué tan grandes son las diversas partes de la figura 5.4. Dados el tamaño del bloque y el número de nodos-i, resulta sencillo determinar el tamaño del mapa de bits de nodos-i y el número de bloques de nodos-i. Por ejemplo, para un bloque de 1K, cada bloque del mapa de bits tiene 1KB (8K bits) y por lo tanto puede llevar el control de 8191 nodos-i (el nodo-i 0 siempre contiene ceros, no se utiliza) para 10000 nodos-i se necesitan dos bloques con mapa de bits. Ya que los nodos-i tienen 32 bytes, un bloque de 1K contiene hasta 32 nodos-i. De acuerdo al ejemplo de la figura 5.4, con 2674 nodos-i, se necesitan 84 bloques de disco para contenerlos a todos.

Más adelante explicaremos con detalle la diferencia que existe entre zonas y bloques, pero por el momento basta con decir que el almacenamiento en disco se puede asignar en unidades (zonas) de 1,2,4, 8 o en general 2 a la **n** bloques. El mapa de bits de la zona lleva el control del almacenamiento libre en zonas (el homónimo de zona en MS-DOS es "clusters"), no en bloques.

Cuando se carga XINIX, el superbloque del dispositivo raíz se lee en una tabla de la memoria. Análogamente, a medida que se montan (más adelante se explican los montajes de SA) los otros SA, sus superbloques también se traen a la memoria. La tabla del superbloque contiene algunos campos que no están presentes en el disco, como el dispositivo del cual provino, los apuntadores a los mapas de bits, el número de nodo-i montado, etc..

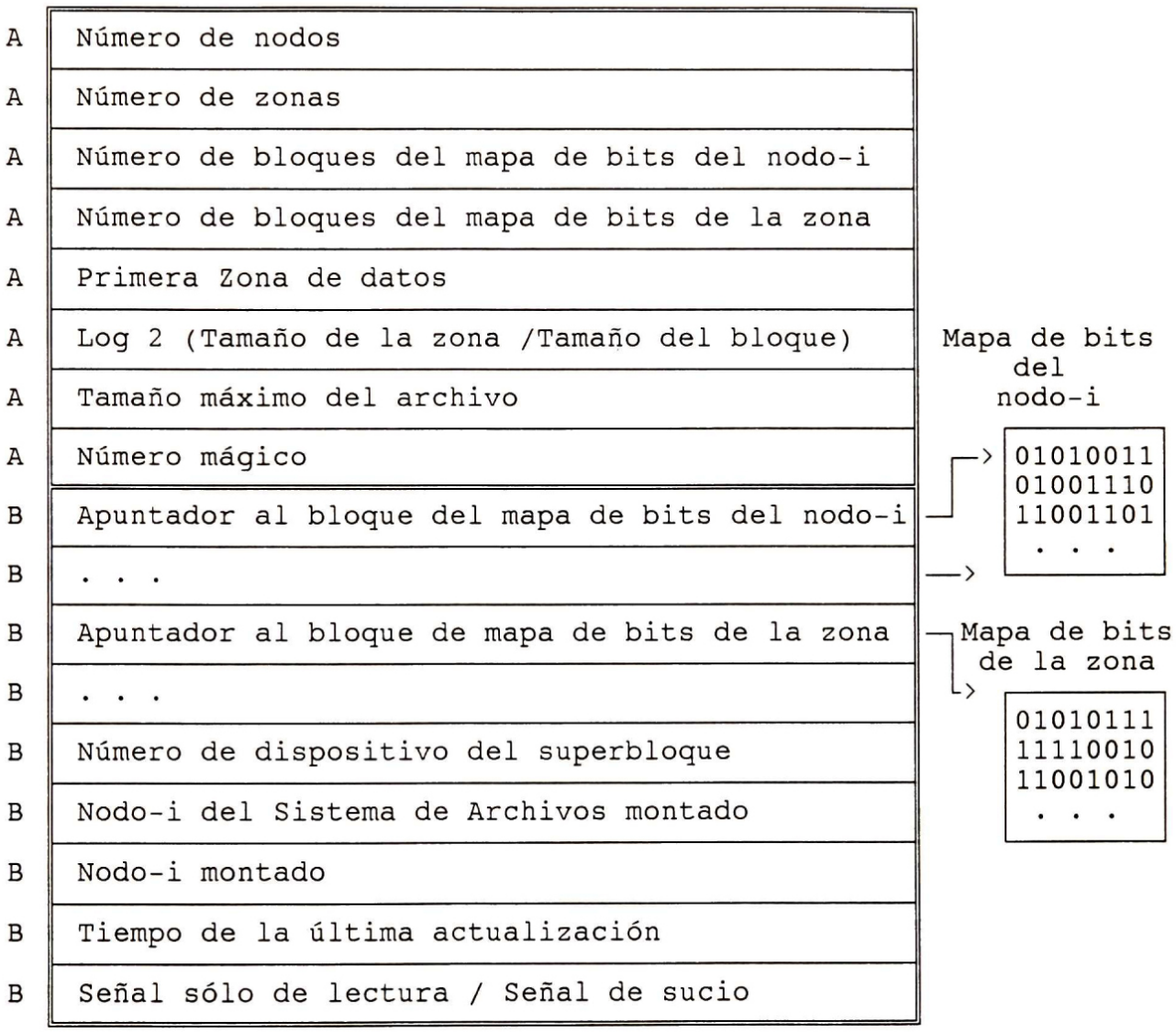


FIGURA 5.5.- SUPERBLOQUE DE MINIX.

5.2.3.- Mapa de bits.

XINIX lleva el control de cuáles nodos-i y zonas están libres mediante el uso de dos mapas de bits (Ver figura 5.5). Cuando el sistema se carga, el superbloque y los mapas de bits del dispositivo raíz se cargan en la memoria. Como se mencionó en un párrafo anterior, la tabla del superbloque contiene algunos campos que no están presentes en el disco. Uno de estos campos es un arreglo cuya k-ésima captación es un apuntador al k-ésimo bloque del mapa de bits del nodo-i (en la memoria).

Cuando un archivo se suprime, es sencillo determinar el bloque del mapa de bits que contiene el bit del nodo-i que se está liberando y hallarlo mediante el arreglo apuntador. Una vez que se halla el bloque, el bit correspondiente al nodo-i liberado se hace igual a cero. Un conjunto de apuntadores semejantes se utiliza con el mapa de bits de la zona.

Al crearse un archivo, el SA busca en los bloques del mapa de bits, uno a la vez, hasta que encuentra un nodo-i libre. Después este nodo-i se asigna para el nuevo archivo. Si todas las ranuras del nodo-i del disco están llenas, la rutina de búsqueda regresa un cero, que es la razón por la cual no se utiliza el nodo-i 0. (Cuando mkfs_hd crea un nuevo SA, anula el nodo-i cero y hace el bit inferior del mapa de bits igual a 1, de manera que el SA nunca intente asignarlo).

Con este antecedente, ahora podemos explicar la diferencia entre los bloques y las zonas. La idea que ocultan las zonas consiste en ayudar a asegurar que los bloques del disco que pertenecen al mismo archivo estén ubicados en el mismo cilindro, con el objeto de mejorar el rendimiento cuando el archivo se lea en orden. El método elegido consiste en hacer posible asignar varios bloques al mismo tiempo. Si por ejemplo, el tamaño del bloque es de 1K y el tamaño de la zona es de 4K, el mapa de bits de la zona lleva el control de las zonas, no de los bloques. Un disco de 20 MB tiene 5K zonas de 4K, por lo tanto 5K bits en su mapa de la zona.

Otra razón para tener zonas tiene que ver con el deseo de conservar las direcciones del disco en 16 bits, principalmente para poder almacenar muchas de ellas en los bloques indirectos. Con un número de zona de 16 bits y una zona de 1K, sólo se pueden direccionar 65K zonas, limitando los disco a 65KB. A medida que los discos crecen, es fácil cambiar a zonas de 2k, 4K sin cambiar el tamaño del bloque (1K). La mayoría de los archivos son menores que 1K, de modo que aumentar el tamaño del bloque significa desperdiciar espacio valioso del disco, fomentando la fragmentación. Desde luego, si se dispone de un disco mayor que 100MB, un tamaño de zona de 2K o 4K no sería tan grave el problema de fragmentación. Por otro lado, se mejoraría la tasa de transmisión de datos, y a su vez, se disminuyen los accesos al disco.

Para la estructura lógica del SA de XINIX se seleccionó un tamaño de zona igual al de bloque, ambos en 1K, debido a que las pruebas se efectuaron con discos duros de 20MB y 40MB.

5.2.4.- Nodos-i.

El esquema del nodo-i de MINIX se da en la figura 5.6, y el nodo-i de XINIX presenta pocos cambios, este se ilustra en la figura 5.7. El nodo-i de XINIX difiere del nodo-i de UNIX de la siguiente manera:

1.- Se almacenan menos apuntadores (9 contra 13).
2.- El campo links se redujo a un byte en XINIX.
3.- El campo gid no se utiliza en XINIX.
4.- El campo de fecha de creación del archivo, no se utiliza en XINIX.

Estos cambios han reducido el tamaño del nodo-i de 64 bytes a 32 bytes, con el objetivo de reducir el espacio en disco y memoria, necesarios para el almacenamiento de nodos-i.

Cuando un archivo se abre, su nodo-i se localiza y se trae a la tabla inode de la memoria, donde permanece hasta que se cierra el archivo. La tabla inode tiene otros campos que no están presentes en el disco (únicamente en memoria), como el dispositivo y número del nodo-i, de manera que el sistema de archivo sepa dónde reescribirlo, en el caso de que se halla modificado mientras está en la memoria. Además tiene un contador por cada nodo-i. Si el mismo archivo se abre más de una vez, sólo una copia del nodo-i se guarda en la memoria, incrementándose el contador cada vez que se abre y disminuyéndose cada vez que se cierra el archivo. Sólo cuando el contador está en cero el nodo-i se suprime de la tabla (y se reescribe en el disco, si se ha modificado).

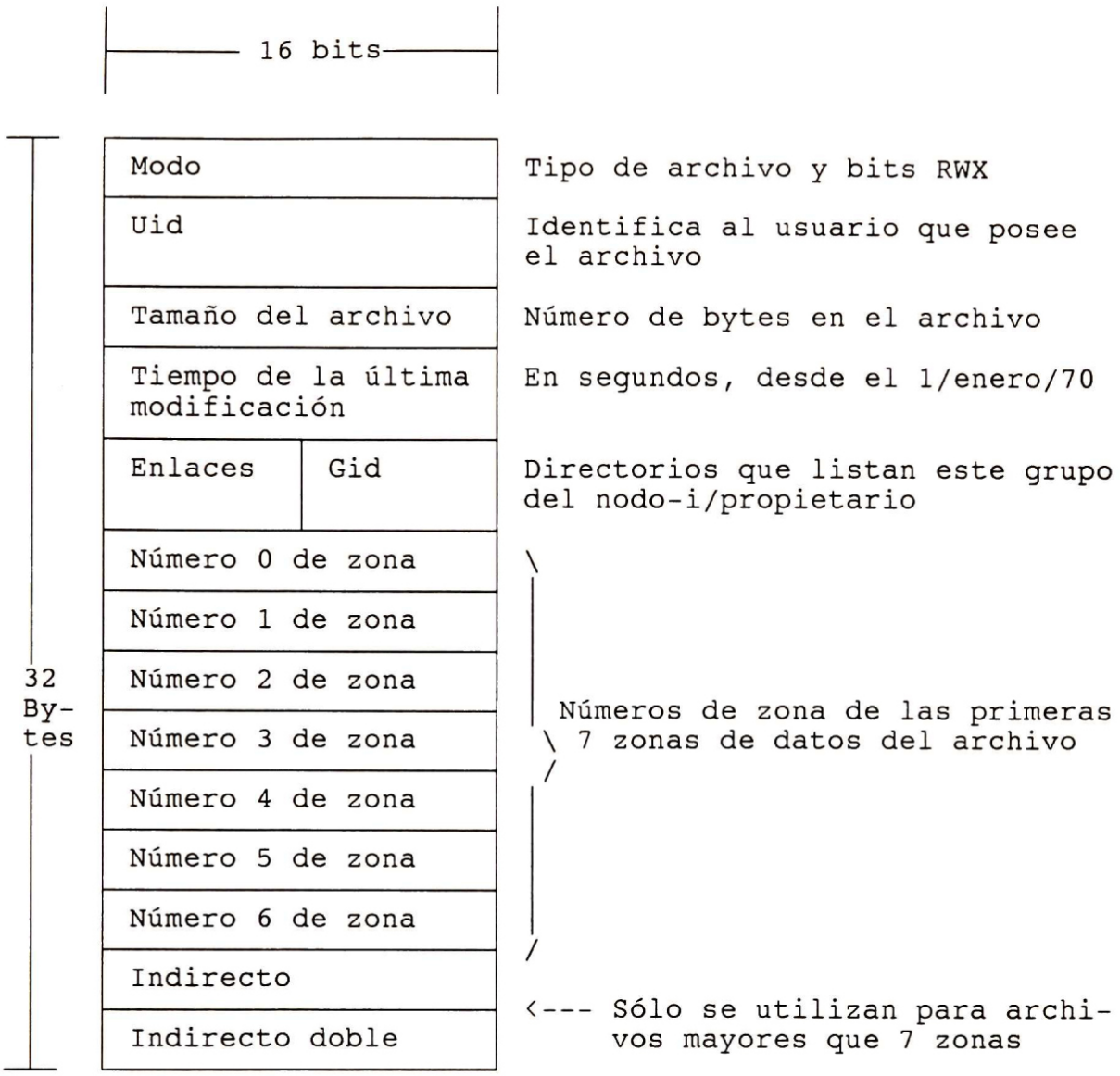


FIGURA 5.6.- El NODO-I DE MINIX.

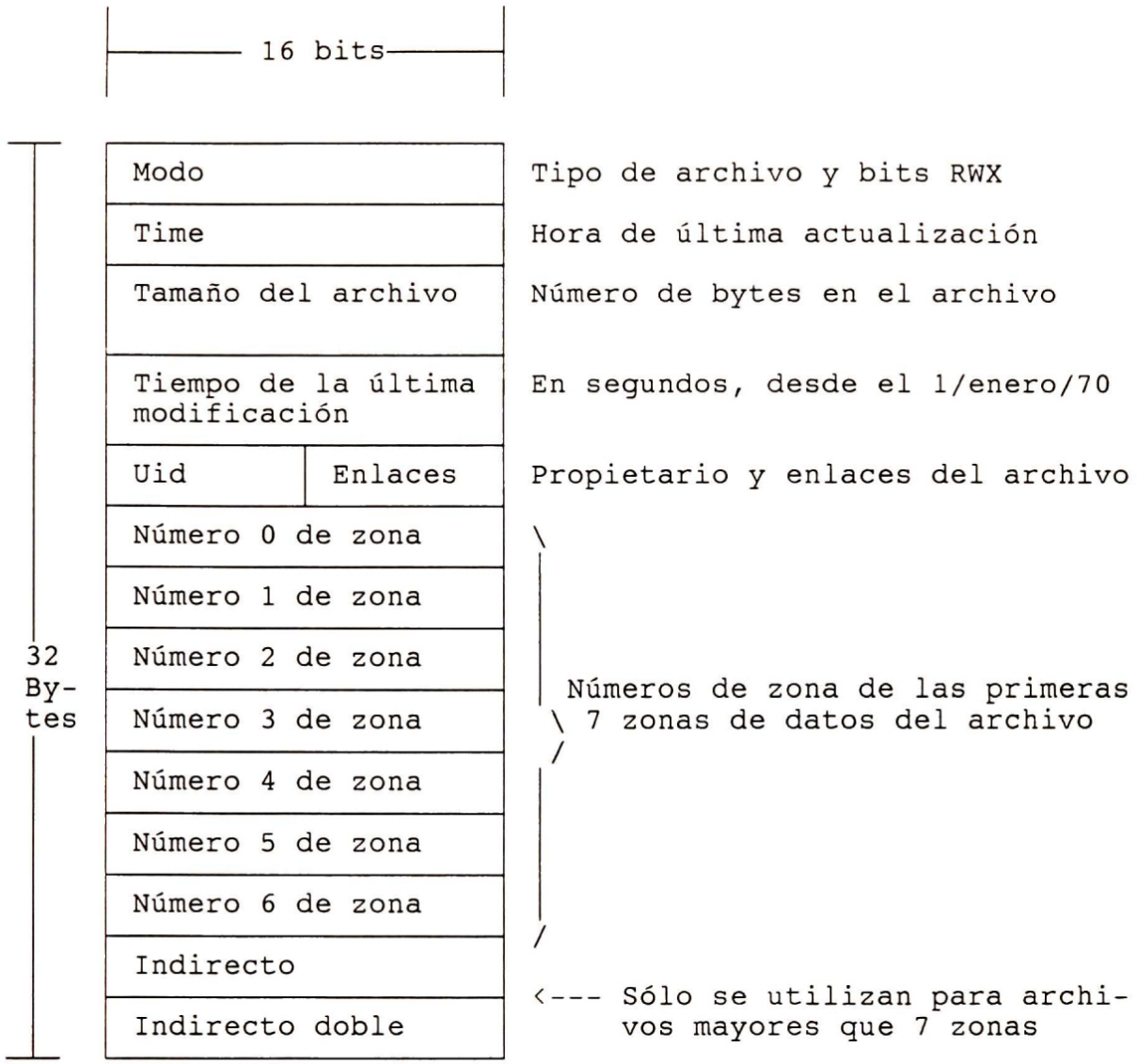


FIGURA 5.7.- El NODO-I DE XINIX.

En la estructura de nodos-i se guardan los nodos-i que están actualmente en uso. En algunos casos se han abierto por medio de llamadas al sistema tales como open() o creat(); en otros casos el sistema de archivo mismo necesita el nodo-i por alguna u otra razón, como la búsqueda de un nombre de trayectoria de un directorio.

La primera parte de la estructura contiene campos presentes en el disco; la segunda parte guarda campos que están presentes en memoria (no en el disco). A continuación se da la estructura inode que utiliza XINIX:

```
struct inode  {
   unshort i_mode; /* tipo de archivo, protección, etc */
   unsigned int i_time; /* tiempo de última actualización
   long i_size; /* Tamaño del archivo actual en bytes*/
   long i_date; /* Cuando se modif. por última vez el ar-
                   chivo */
   char i_uid; /* id. del usuario propietario del archivo
   char i_links; /* cuantos enlaces para este archivo*/
   unshort i_zone[NR_ZONE_NUMS]; /* Num. de zonas para di-
               recto, indirecto y doble indirecto */

   /* Los sig. campos no aparecen en el disco */
   unshort i_dev; /* q' disp. se encuentra sobre el
                     nodo-i*/
   unshort i_num; /* Num. de nodo-i sobre su disp. menor*/
   short int i_count; /* #de veces que es usado el nodo-i;
                      0 significa que la ranura está libre*/
   char i_dirt; /* Clean o Dirty */
   char i_pipe; /* colocar a i_pipe si es pipe */
   char i_seek; /* colocado en LSEEK, eliminado en READ/
                   WRITE */
};
```

El principal propósito del nodo-i de un archivo consiste en indicar dónde están los bloques de datos. Los primeros 7 números de zona se dan en el nodo-i mismo. Con bloques y zonas de 1K, los archivos hasta de 7k, no necesitan bloques indirectos. Después de 7K se necesitan zonas indirectas. Con bloques y zonas de 1K y número de zona de 16 bits, un bloque indirecto sencillo contiene 512 entradas, lo que da 524KB de almacenamiento. Y un bloque indirecto doble apunta a 512 bloques indirectos sencillos, lo cual da 256MB. (En realidad este límite no es alcanzable, porque con número de zona de 16 bits y zonas de 1K, sólo se pueden direccionar zonas de 64K, las cuales tienen 64 MB; para un disco de mayor capacidad tendríamos que recurrir a una zona de 2K).

5.2.5.- Memoria Caché.

La memoria caché se implementó en base a la necesidad de reducir la frecuencia de accesos a disco y, a su vez, mantener la mayor información posible de disco en memoria [2].

La memoria caché es una estructura en memoria que se crea cuando se inicializa XINIX. Consta de 30 buffers, cada uno de estos buffers se enlazan en una lista doblemente ligada.

Cabe mencionar que cada buffer está formado de dos partes: un encabezado que identifica al buffer y un arreglo del mismo tamaño que los bloques del disco (1024 Bytes).

5.2.6.- Montaje y Desmontaje del Sistema de Archivo.

Después de inicializar la memoria caché se crea el directorio raíz de XINIX, con los subdirectorios dev, usr_a, usr_c; las entradas del subdirectorio dev identifican a los manejadores de disco como fd0 y hd1 (A: y C: en MS-DOS), mientras que los directorios usr_a y usr_c están vacíos, no se puede ni debe crearse nada en ellos, se usan para montar sistemas de archivo de disco, el montaje sirve para poder hacer referencia a sus archivos y directorios. El montaje se lleva a cabo con el comando **mount**; la sintaxis del comando es la siguiente:

mount ident_manejador directorio_vacío

ejemplo:

```
CONSOLE>> mount /dev/hd1 /usr_c
```

indica que deseamos montar el sistema de archivo del disco duro, en el subdirectorio usr_c. De esta manera, se mapea el directorio raíz del sistema de archivo de disco con el subdirectorio usr_c, ahora se tiene acceso a cualquier archivo o subdirectorio del disco duro.

Un archivo en un subdirectorio puede ser accesado usando trayectorias absolutas o relativas. Al usar trayectorias absolutas accesamos un archivo desde cualquier otro directorio en el sistema de archivo, por ejemplo:

/usr_c/dir/ . . . /file

Esto es posible debido al campo s_isup (de la estructura del superbloque) que apunta al nodo-i del directorio sobre el que se hizo el montaje. Usando ambos es como se pueden utilizar las trayectorias absolutas.

Por otra parte, el comando **umount** desmonta el sistema de archivo, que fué montado en el nombre de dispositivo, ejem.:

umount nombre_de_dispositivo

```
CONSOLE>> umount /dev/hd1
```

La labor que hace este comando consiste en eliminar la referencia entre el directorio donde se montó y el sistema de archivo del disco.

5.3.- Cambios al Manejador de Disco Duro para Manejar el NSAX

El manejador de disco duro está implementado en el archivo x2win.c; en la función w_do_rdwt() se toma el número de sector lógico de la petición en turno (del campo drdba, de la estructura dreq, mostrada en la sección 3.3.1), a ese número de sector se multiplica por 2 para obtener el bloque de 1024 Bytes que maneja el NSAX.

Hay que recordar que el manejador de disco maneja sectores físicos de 512 Bytes, y para cualquier acceso de XINIX al disco, lo hace utilizando 2 sectores contiguos de 512 Bytes.

El cambio efectuado al manejador de disco duro para el NSAX consiste únicamente en el aumento del bloque a 1024 Bytes.

5.4.- Cómo se Instala el NSAX en el Disco Duro.

En al sección 5.2.2 se dió una breve explicación del procedimiento que se realiza para construir el NSAX en el disco duro. Nuevamente, recordemos que la segunda partición del disco debe estar disponible para el SO XINIX, y también debemos conocer el tamaño en número de bloques (de 1024 Bytes) de esa partición. Si se desconoce o se desea saber ese valor, se recomienda utilizar la utiliería NU de NORTON y localizar las direcciones absolutas en el disco duro de cilindro, sector y cabeza, así como se indica a continuación:

```
cilindro - 0
sector   - 1
cabeza   - 0
```

dando esos valores encontramos el lugar físico donde se localiza la tabla de particiones (Observe la figura 4.8) y, de esta manera, podemos obtener el tamaño de la segunda partición en sectores. El usuario de XINIX debe notar que el tamaño de la segunda partición se da en sectores de 512 Bytes y necesitamos convertirlo a bloques de 1024 Bytes.

Otro punto importante que debemos considerar es el número máximo de bloques que soporta el programa (mkfs.c): es de 24,552 (25MB). Es decir, la segunda partición del disco duro no debe exceder los 24552 bloques (de 1024 Bytes).

Ahora, en el caso de que el usuario de XINIX proporcione un número de bloque mayor al que tiene asignada esa partición, no sucede nada, únicamente no se efectúa el comando y se regresa al "prompt" de CONSOLE>> y además el usuario obtiene en el monitor la información acerca del número de bloques (de 1024Bytes) que se tienen disponibles en la segunda partición, ésta es la forma más fácil y segura de determinar el tamaño de la segunda partición. El programa mkfs.c contempla esta posibilidad, y protege la integridad de la información del disco en el caso de dar un número de bloque más allá de los que tiene asignada la segunda partición.

En resumen, para instalar el NSAX en disco duro, únicamente se hace uso del comando **mkfs_hd** de la siguiente manera:

```
CONSOLE>> mkfs_hd c: número_de_bloques
```

Al finalizar el formato lógico, el siguiente paso consiste en montar el sistema de archivo de la siguiente manera:

```
CONSOLE>> mount /dev/hd1 /usr_c
```

en donde /dev/hd1 es la identificación lógica del manejador de disco duro y /usr_c se utiliza para montar el sistema de archivo de disco duro.

Una vez montado el sistema de archivo, el usuario de XINIX se cambia al directorio, digamos, de /usr_c, por medio del comando **cd** ("change dir")

```
CONSOLE>> cd /usr_c
```

En el párrafo siguiente explicaremos cómo funciona el programa mkfs.c, el cual ya se mencionó, es el encargado de construir el NSAX en el disco duro.

5.4.1.- Descripción del Programa mkfs.c

Al utilizar el comando mkfs_hd, el usuario proporciona el parámetro número de bloques. A partir del número de bloque obtenemos el número de nodos-i necesarios para el NSAX, de la siguiente manera:

```
num_nodos-i = Num_bloques/3 + 8
```

La función mkfshd() es la rutina principal del programa mkfs.c, ésta recibe el número de bloques y calcula el número de nodos-i. Siempre un ejemplo aclara cualquier duda, si el número de bloques disponibles es de 8000, entones el número de nodos-i será de 2674.

El programa utiliza una memoria caché de 20 bloques de 1024 Bytes cada uno, su estructura se indica a continuación:

```
struct inode  {
  char blockbuf[BLOCK_SIZE];
  int blocknum;
  int dirty;
  int usecnt;
} cache[CACHE_SIZE];
```

El propósito de la memoria caché, es el de acelerar el proceso de construcción del NSAX.

Continuando con la descripción de la función mkfshd(); se inicializa la memoria caché. Enseguida se emplea un buffer de 1024 Bytes y se llena de "ceros". Este buffer se pasa como parámetro a la función write_block(), junto con el número de bloque: "cero" (write_block(0,buffer)). Como la intención es empezar a construir el NSAX (Véase la figura 5.4), el primer bloque que se escribe al disco, es el bloque de carga (boot),

o sea, el bloque "cero". En la figura 5.8 se muestra el primer bloque del NSAX, es decir, el bloque de carga, y queda reservado para uso futuro.

Ahora bien, anteriormente se dijo que el tamaño del bloque y el tamaño de la zona son ambos de 1KB. De ahora en adelante, trabajaremos con zonas, debido a que la estructura del superbloque maneja el campo s_nzones, al cual se le asigna el número de zonas.

Regresando al programa, ahora, aplicando la función super(), la cual tiene dos parámetros: el número de zona y el número de nodos-i. Con estos dos valores se calcula el resto de los campos de la estructura super_block, y de esta forma se delimita el número de zonas asignadas a cada parte del sistema de archivo (tales como, el número de zonas para el mapa de bits de nodos-i, el número de zonas para el mapa de bits de datos, la primera zona de datos, etc.), observe la figura 5.4.

Toda la información del superbloque se guarda en un buffer y se escribe en el disco empleando la función write_block(1,buffer)(Vea la figura 5.9). Por supuesto, hasta el momento llevamos dos zonas grabadas en el disco, el bloque de carga y el superbloque.

Casi al final de la función super(), se procede a poner ceros (indicando bits disponibles) en el mapa de bits de nodos-i (es el tercer bloque, se utiliza insert_bit()) hasta el número de nodos-i que se obtuvo. Por ejemplo, 2674 nodos-i ocupan 167 enteros en el mapa de bits de nodos-i y el resto del bloque se "llena" con "unos", señalando que no existen o están ocupados, referirse a la figura 5.10.

De la misma manera, para el bloque de mapas de bits de zonas, primero se obtiene el número de zonas que se va a destinar para datos, así como se indica:

```
long_map_zonas = nrzones - s_firstdatazone + 1
```

si nrzones es igual a 8000 y s_firstdatazone es igual a 88, entonces long_map_zonas es igual a 7913. Por lo tanto, se necesitan 7913 bits en el bloque del mapa de bits de zonas para representar a 7913 zonas. Los primeros 7913 bits se marcan con "ceros", indicando que están disponibles y el resto, es decir 279 bits, se señalan con "unos", marcándolos como no existentes. En la figura 5.11 se muestra el mapa de bits de las zonas.

FIGURA 5.8.- BLOQUE DE "BOOT" DEL DISCO DURO, PRIMER SECTOR FISICO DONDE EMPIEZA EL SISTEMA DE ARCHIVO DE XINIX. ESTE SECTOR DE 1024 BYTES QUEDA RESERVADO PARA USO FUTURO.

Side 0, Cylinder 528, Sector 1

```
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000   PRIMER SECTOR
00000000 00000000 00000000 00000000 00000000 00000000   DE 512 BYTES
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
```

Side 0, Cylinder 528, Sector 2

```
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000   SEGUNDO SECTOR
00000000 00000000 00000000 00000000 00000000 00000000   DE 512 BYTES
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
```

FIGURA 5.9.- PRIMER SECTOR DE 512 BYTES DEL SUPER BLOQUE.

Side 0, Cylinder 528, Sector 3

```
720A401F 01000100 58000000 001C0810 7F134006 5D0283C4
06EB6A8B C60BC074 0C3D0100 74213D02 007436EB 4B8B46FE
BA0A00F7 E28946FE C45E0626 8A079805 D0FF0146 FEEB318B
46FC0000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
```

0A72 = 2674 Nodos-i
1F40 = 8000 Zonas
0001 = 1 bloque mapa de bits de nodos-i
0001 = 1 bloque mapa de bits de zonas o datos.
0058 = 88 Primer zon de datos
0000 = s_log_zone_size
10081C00 = 268'96691 s_max_size=268
137F = Núm. mágico

SEGUNDO SECTOR DE 512 BYTES DEL SUPER BLOQUE

Side 0, Cylinder 528, Sector 4

```
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
```

FIGURA 5.10.- MAPA DE BITS DE NODOS-I

```
= Side 0, Cylinder 528, Sector 5 ═══════════════════════════
FFFFFF07 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 0000F8FF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF
```

```
Bit - 0 Bit libre
Bit - 1 Bit ocupado

Num_nodi=Num_blo/3+8
        =8000/3+8
        =2674

             167 entero
         ┌────────────
 16 bits │2674Nodosi
```

En este ejemplo tenemos 167 enteros que pueden representar a 2674 nodos

```
= Side 0, Cylinder 528, Sector 6 ═══════════════════════════
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF
```

FIGURA 5.11.- MAPA DE BITS DEL BLOQUE DE ZONAS (O DATOS).

```
Side 0, Cylinder 528, Sector 7
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF1F 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
```

1 Bloque de 1024 B
tiene 8192 bits.

Cada bit "uno" ind
ca un Bloque en la
zona de datos.

```
Side 0, Cylinder 528, Sector 8
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00FEFFFF   <—— Se marcan com
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF       no existentes
FFFFFFFF FFFFFFFF
```

279 bits no están disponibles, por lo tanto, 8192-279 = 7913 bits disponi-
bles. Con este Bloque se pueden direccionar hasta 7913 Bloques de datos.

FIGURA 5.12.- PRIMER BLOQUE DE NODOS-I.

```
 Side 0, Cylinder 528, Sector 9

FF413116 70010000 C7077911 02045800 00000000 00000000 41FF = i_mode
00000000 00000000 F6811817 04030000 C7075241 00015900 1631 = i_time
00000000 00000000 00000000 00000000 FF411917 20000000 00000170 = i_size
C7075241 00025A00 00000000 00000000 00000000 00000000 07C71179 = i_date
F6812C16 16020000 BC071120 00015B00 00000000 00000000 02 = i_uid
00000000 00000000 F6811814 2E2E0000 C7073300 00015C00 04 = i_links
5D005E00 5F006000 61006200 64000000 00000000 00000000 0058 = i_zone[0]
00000000 00000000 00000000 00000000 00000000 00000000        i_zone[1]
00000000 00000000 00000000 00000000 00000000 00000000        i_zone[2]
00000000 00000000 F6813816 49260000 BC071120 00016900        i_zone[3]
6A006B00 6C006D00 6E006F00 71000000 F6811413 86160000        i_zone[4]
BC071120 00017400 75007600 77007800 79000000 00000000        i_zone[5]
00000000 00000000 00000000 00000000 00000000 00000000        i_zone[6]
00000000 00000000 F6811713 AB050000 BC071120 00017A00        i_zone[7]
96000000 00000000 00000000 00000000 F6813810 86160000        i_zone[8]
C707F410 00017B00 90009100 92009300 94000000 00000000
F6813714 F1480000 C707C320 00017C00 7D007E00 7F008000
81008200 84000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 F6811A0A 2E010000
C7076411 00019500 00000000 00000000 00000000 00000000
4980080A 9A0E0000 BC071120 00019700 98009900 9A000000
00000000 00000000
```

```
 Side 0, Cylinder 528, Sector 10

F681170A 79260000 C7076411 00019B00 9C009D00 9E009F00
A000A100 A3000000 F6811B0A D64F0000 C7076411 0001A600
A700A800 A900AA00 AB00AC00 AE000000 F6812B0A 55160000
C7076411 0001BB00 BC00BD00 BE00BF00 C0000000 00000000
F6811A15 7F830000 BC071120 0001C100 C200C300 C400C500
C600C700 C9000000 FF411016 20000000 BC071120 0002E300
00000000 00000000 00000000 00000000 F6811015 34000000
BC071120 0001E400 00000000 00000000 00000000 00000000
F6810316 D8000000 BC071120 0001E500 00000000 00000000
00000000 00000000 F6810516 82030000 BC071120 0001E600
00000000 00000000 00000000 00000000 F6812F16 472A0000
BC071120 0001E700 E800E900 EA00EB00 EC00ED00 EF000000
F6813116 7A010000 C7077911 0001F300 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
```

32 Nodos-i por cada bloque de 1024 Bytes.

FIGURA 5.13.- BLOQUE DIRECTORIO, PRIMER ZONA DE DATOS.

0001 = Número de nodo-i
002E = "." es el nombre del directorio (Se puede utilizar 14 bytes para el nombre del directorio)

```
╔ Side 1, Cylinder 529, Sector 22 ═══════════════════ Hex format
║01002E00 00000000 00000000 00000000 01002E2E 00000000................
║00000000 00000000 02007872 67000000 00000000 00000000 ......xrg.......
║03006469 72310000 00000000 00000000 04007265 73703100 .dir1.....resp1.
║00000000 00000000 05007872 66732E63 00000000 00000000 .......xrfs.c...
║08007869 6E69782E 63000000 00000000 09007864 69736B2E xinix.c...xdisk.
║68000000 00000000 0B007863 6F6E662E 63000000 00000000 h.....xconf.c...
║0C006469 736B0000 00000000 00000000 0D007832 72656C6F .disk..........
║63000000 00000000 0F007869 6F2E6800 00000000 00000000 .......xio.h....
║1000656A 312E6578 65000000 00000000 11007869 6E69782E ej1.exe...xinix.
║66000000 00000000 12006D6B 66732E63 00000000 00000000 f.....mkfs.c....
║13007832 686C696F 2E630000 00000000 14007832 77696E2E x2hlio.c..x2win.
║63000000 00000000 15006469 72320000 00000000 00000000 c...dir2........
║16006669 6C650000 00000000 00000000 17007475 312E6300 .file.....tu1.c.
║00000000 00000000 18006A00 00000000 00000000 00000000 .......j.......
║19007072 742E6300 00000000 00000000 1A006669 6C653200 prt.c.....file2.
║00000000 00000000 00000000 00000000 00000000 00000000 ................
║00000000 00000000 00000000 00000000 00000000 00000000 ................
║00000000 00000000 00000000 00000000 00000000 00000000 ................
║00000000 00000000 00000000 00000000 00000000 00000000 ................
║00000000 00000000 00000000 00000000 00000000 00000000 ................
║00000000 00000000 00000000 00000000 00000000 00000000 ...............
║00000000 00000000

╔ Side 1, Cylinder 529, Sector 23 ═══════════════════
║
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000 00000000 00000000 00000000 00000000
║00000000 00000000
```

Para finalizar con super(), el último procedimiento consiste en colocar un bit "uno" al mapa de bits de nodos-i y otro al mapa de bits de las zonas, nos estamos refiriendo al al primer bit (el cero), de esta manera quedan reservados y el sistema de archivo ya no tendrá acceso a esos dos bits.

Después de registrar en el disco duro los primeros cuatro bloques (Vea la figura 4.5, bloques 0,1,2 y 3), corresponde ahora llevar a cabo el asignamiento (asigna_inode()) de nodos-i. Primero lee el bloque 4 (read_block(4,inode)) lo mantiene en un buffer y se le coloca el modo (i_mode, protección) y la identificación del usuario, enseguida, vuelve a escribir el mismo bloque, pero ya modificado (write_block(4,-inode)) e inserta en el mapa de bits de nodos-i el número de nodo-i correspondiente.

A continuación, procede a instalar el directorio raíz (rootdir()). El número de nodo-i del párrafo anterior lo toma como parámetro: rootdir(nodo-i). En rootdir() el primer paso consiste en asignar la zona (asigna_zone()), sería la primer zona de datos, para el ejemplo que iniciamos al empezar esta sección, la zona de datos empieza en el bloque 88. Se limpia el bloque 88 al grabar un buffer lleno de "ceros", de esta manera, queda disponible el área de directorios. Y posteriormente inserta el bit uno en el mapa de bits de las zonas y regresa el número de zona 88.

El número de zona 88 lo captura add_zone(). El propósito de add_zone() consiste en leer el bloque de nodos-i, y actualizar los campos de fecha y tamaño de directorio, así como asignar el número de zona (88) al campo i_zone[0] de la estructura inode y vuelve a escribir el bloque de nodos-i al disco. En la figura 5.12 se indica la estructura completa de los nodos-i, representada en notación hexadecimal, los primeros 16 enteros corresponden al directorio raíz.

El paso siguiente consiste en grabar el nombre del directorio actual '.' y padre '..', en el bloque 88 (bloque directorio) por medio de la función enter_dir(). Aquí se utiliza la estructura de directorios que se mostró en la figura 5.3. En la figura 5.13 observamos que los primeros 32 bytes corresponden a los directorios '.' y '..' que originalmente quedan grabados en el disco.

Y por último se incrementa la cuenta de las "ligas", es decir, el campo i_nlinks de la estructura inode.

5.5.- Copiando un Archivo de MS-DOS a XINIX.

El SO original de XINIX tiene implementado un comando llamado **cdxf**, con este comando el usuario copia un archivo desde un diskette con formato de MS-DOS a otro diskette con el formato de XINIX. Pues bien, ahora con el NSAX el usuario puede copiar un archivo de la primera partición del disco duro a la segunda partición donde se encuentra instalado XINIX. La manera de llevar a cabo esta operación es por medio del comando **cdxf_hd**, enseguida se describe su aplicación:

```
CONSOLE>> cdxf_hd subdir_de_DOS nom_arch_DOS nom_arch_XINIX
```

En la sección 5.5.2 se explica como el programa cdxf_hard_disk() (en x2win.c) ejecuta la copia de archivos.

Otro comando no menos importante, implementado para el disco duro es **dir_hd**. Con este comando el usuario de XINIX puede ver el contenido del directorio raíz de MS-DOS.

En la sección 5.5.1 se describe el funcionamiento de la función dirmsdos_hd(), la cual es la que lista el directorio raíz de MS-DOS y obviamente el comando dir_hd ejecuta esta función.

5.5.1.- Descripción del Comando dir_hd.

El función del comando **dir_hd** es similar al **dir** de MS-DOS, únicamente se teclea dir_hd y obtenemos la lista de archivos y directorios del directorio raíz de MS-DOS.

Ahora se explica la función dirmsdos_hd(), la cual hace posible la tarea del comando dir_hd.

Anteriormente se mencionó que XINIX trabaja con bloques de 1024 bytes, pues bien, al hacer uso del comando dir_hd (esto también se aplica para el comando cdxf_hd) nos vemos en la necesidad de cambiar el tamaño del bloque a 512 bytes. Esto tiene una explicación, ya que el uso del comando dir_hd implica invadir la partición de MS-DOS (la primera partición) resulta más práctico trabajar con bloques de 512 bytes, en esa partición, que con bloques de 1024 bytes.

Esta modificación de bloques de 1024 a 512 o viceversa, la controla la función w_do_rdwt(). Entonces, cuando se usa el comando dir_hd, el programa cambia el tamaño del bloque a 512 bytes y además podemos entrar a la partición de MS-DOS. Recuérdese que el usuario de XINIX únicamente puede trabajar en la segunada partición y no debe invadir la partición de MS-DOS, por supuesto, a excepción del empleo de los comandos dir_hd y cdxf_hd.

Continuando con la función dirmsdos_hd(), la primer tarea consiste en leer el sector "boot" (Vea la figura 4.6, cilindro 0, cabeza 1, sector 1) y obtener así toda la información del sector boot que se muestra en la figura 3.5. Enseguida se comprueba si el byte 21 (Byte descriptor del medio) corresponde a la identificación del disco duro (0xF8, para el disco duro). De cumplirse esa condición, se procede a leer el área de directorios y se guarda en un buffer, y posteriormente se lista el contenido del buffer e identifica a los archivos y a los directorios. Y finalmente se da el número de directorios y archivos que contiene el directorio raíz de MS-DOS.

5.5.2.- Descripción del Comando cdxf_hd.

La función cdxf_hard_disk() lleva el control del comando **cdxf_hd**.

Antes de empezar debemos recordar que la FAT se ha incrementado poco a poco con la aparición de unidades de disco duro cada vez más grandes. No obstante, el NSAX se instaló en un disco de 20 MBytes el cual maneja una FAT de 12 bits, así también, se instaló el NSAX en un disco duro de 40 MBytes con una FAT de 16 bits. Teniendo en mente lo anterior, la función cdxf_hard_disk(), comprueba primero en la tabla de particiones (Vea la tabla 4.2) el cuarto byte que tiene por nombre 'tipo de partición'. Si dicho byte es 01H entonces la FAT es de 12 bits y si es 04H, entonces la FAT es de 16 bits.

De cualquier forma, primero se comprueba qué FAT tenemos en el disco y después se procede a efectuar lo pertinente en cada caso.

Continuando se lee el sector "boot" de MS-DOS (Vea la figura 3.5) y se obtiene la información de cómo se formateó el disco, también se obtiene el número de sectores por cluster, el número de copias por FAT, el número de sectores asignados para el área de directorios, etc. Toda esta información se guarda en varias variables.

Ahora obtenemos los nombres del directorio, del archivo del MS-DOS y del archivo que se va a crear en XINIX.

A continuación leemos el área de directorios de MS-DOS (En la figura 4.6 se señala el área de directorios) y se obtiene el cluster inicial del subdirectorio solicitado. De acuerdo con la estructura de directorios y archivos de MS-DOS. Vea la figura 3.7.

Posteriormente se lee el área de archivos del subdirectorio, hasta encontrar el archivo que se va a copiar a XINIX y se obtiene el cluster inicial de ese archivo.

Una vez que se localizó el archivo, éste se guarda en un buffer del tamaño del archivo. Y después se crea el archivo en la partición de XINIX, vaciando el buffer anterior.

Referencias en Bibliografía:

[6] Tanenbaum, 1988.
[2] Tesis de Ruth Delgado, 1990.

CAPITULO 6

MANEJADOR DE IMPRESORA PARA XINIX.

6.1.- Organización Física.

Las pruebas del manejador de la impresora se realizaron en una impresora de matriz de puntos de impacto (Citizen 120D equivalente a Epson FX) con cabeza de impresión de 9 "pines".
La impresora se compone principalmente de las siguientes partes:

a) El mecanismo que mueve el papel, e incluye:
   - La circuitería que activa el motor de pasos.
   - El tractor que alimenta el papel.
   - El rodillo que da fricción al papel.
b) El mecamismo que mueve la cabeza de impresión y el que hace girar la cinta de impresión.
c) Y la cabeza de matriz de puntos.

Veamos cómo funciona la cabeza de impresión. Todas las impresoras de matriz de puntos operan bajo el mismo principio básico de alambres o "pines" que golpean la cinta. Cada pin imprime un punto. Activando los pines en diferentes patrones, la impresora genera caracteres (letras) o imágenes. La impresora de matriz de puntos usa la misma tecnología que encontramos en cualquier bocina. Todas las bocinas trabajan empujando o jalando una pieza circular de cartón. La cabeza de la impresora empuja y jala un alambre pequeño. Las componentes necesarias para acompañar este movimiento, consiste de un magneto y una bobina de alambre. Cuando se energiza la bobina la corriente genera un campo magnético que repele el campo magético del magneto permanente. Estas fuerzas repelen el magneto permanente originando el movimiento. El alambre unido al magneto permanente se mueve hacia afuera, golpea la cinta, y la empuja contra el papel. Un resorte jala el magneto permanente y regresa el alambre a su posición original [13].

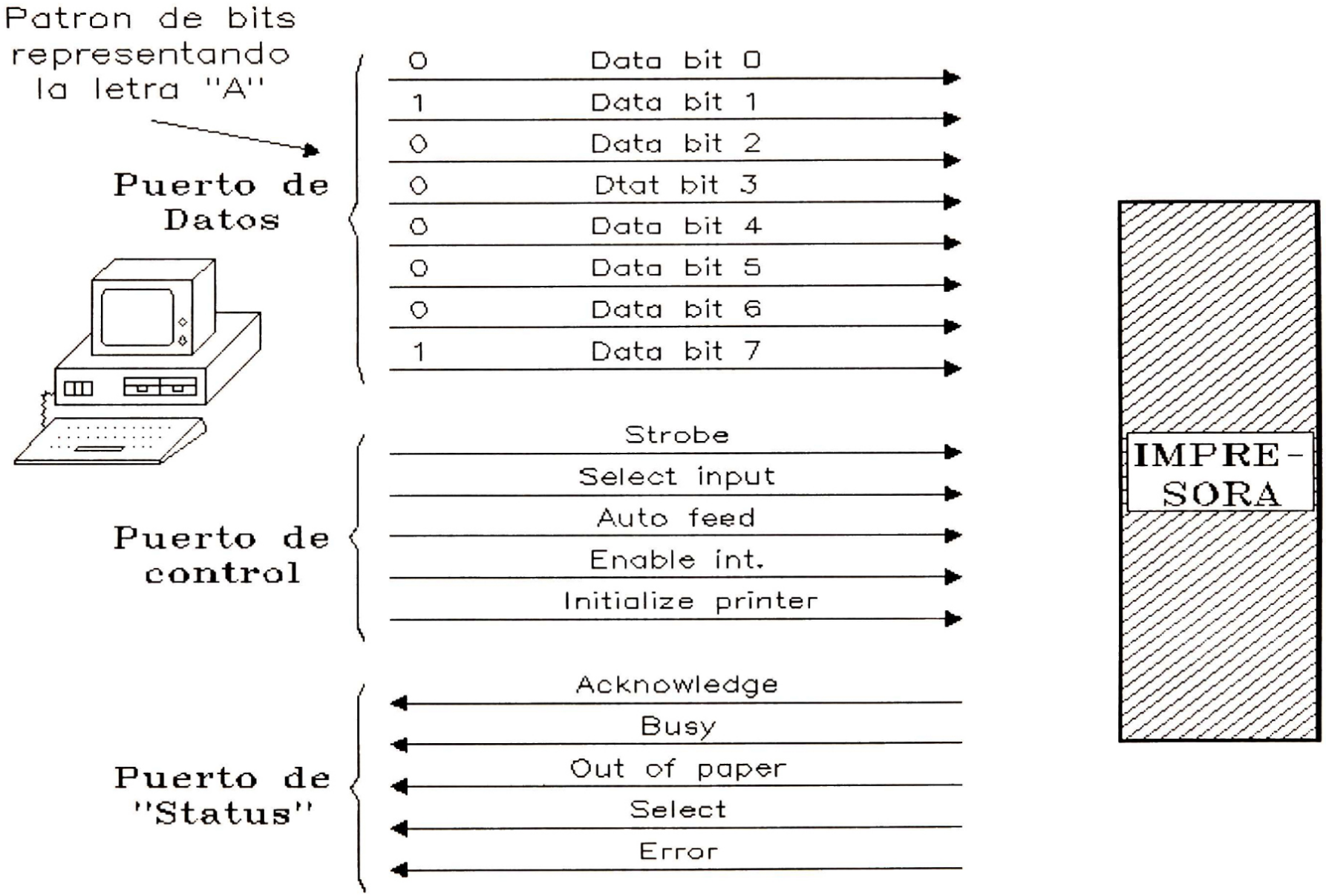
6.1.1.- Cómo Funciona la Impresora.

Las PC's transmiten datos a la impresora. La circuitería de la impresora debe ser capaz de determinar cuando se va a transferir la información. Algunas veces la impresora tuvo que reconocer que el dato fué transferido correctamente. La circuitería de la impresora recibe el dato desde la PC a través de una interfaz paralela.

6.1.1.1.- Interfaz Paralela.

Con una interfaz paralela, cada bit que representa un caracter o instrucción tiene su propio alambre. Por ejemplo, el patrón de bits para representar la letra "A" es 0100 0001. Para transferir la letra A, se necesitan por lo menos 8

FIGURA 6.1.- INTERFAZ PARALELA



alambres por cada uno de los bits que respresentan la letra A. En la figura 6.1 se ilustra la interfaz paralela y el ejemplo expuesto.

Se necesitan dos alambres más para decirle a la impresora cuando busque por un dato y para habilitar la impresora para que reconozca la repecepción de un dato. Estas señales son **strobe** y **acknowledge**, respectivamente.

Además de esas dos señales existen otras que inicilizan la impresora: **select** e **initialize printer**.

También la impresora cuenta con señales que manejan las condiciones de error que puedan presentarse, tales como: **busy out of paper**, **error**, etc.

6.1.1.2.- Almacenando Datos en la Memoria de la Impresora.

Los datos que llegan a la impresora son almacenados en memoria RAM ("random-access-memory"). La memoria RAM puede ser desde 2K o más grande que 5MBytes. Aunque el dato más común que llega a la impresora es texto, el dato también puede incluir:

- Imágenes de qué es lo que se va a imprimir.
- Programas

La información que todavía no es procesada se guarda en la memoria RAM de la impresora, a esta se le llama **buffer**. La RAM también es usada para almacenar instrucciones sobre cómo dibujar las letras u objetos.

Cuando se envía un programa a la impresora, se crea una imágen en la RAM de la impresora. La imágen puede ser una línea o un dibujo complicado, o la imágen puede representar una hoja entera de papel o varias hojas.

6.2.- Organización Lógica.

6.2.1.- Cómo Imprime.

La PC puede estar conectada a una impresora con una interfaz paralela estándar. Vea la figura 6.1. El monitor monocromático y el adaptador de la impresora paralela puede ser usada para este propósito. Las señales de la interfaz de la impresora son mostradas en la misma figura 6.1.

La operación de la interfaz paralela es sencilla. La impresora no acepta cualquier entrada a menos de que la línea de "select input" esté en el estado apropiado. La línea "initialize printer" es usada para inicializar la impresora cuando el sistema se enciende. La señal correcta debe estar por lo menos 50 micro segundos.

El dato que va a ser impreso es puesto en las 8 lineas de datos "data bit". Entonces la línea strobe es pulsada. La impresora procesa el dato y entonces envía un pulso de

regreso sobre la línea "acknowledge". Cuando el pulso de reconocimiento "acknowledge" es recibido por la computadora, otro caracter puede ser enviado a la impresora. En lugar de esperar por el pulso de reconocimiento, la computadora puede comprobar la línea de ocupado ("busy"). La computadora puede enviar tantos caracteres a la impresora mientras ésta no esté ocupada. La entrada "out of paper" le dice a la PC que la impresora no tiene papel. La entrada "select" le dice a la PC que la impresora está 'en linea'. La entrada "error" le dice a la PC que la impresora está fuera de línea, no hay papel, o existe otro estado de error. El estado "busy" también refleja estas condiciones de error.

El microprocesador 8086 puede introducir o sacar datos para controlar o leer las líneas de interfaz de la impresora paralela.

Las direcciones de entrada/salida para el adaptador de la impresora paralela se muestran en la Tabla 6.1.

TABLA 6.1

FUNCION	DIRECCIONES DE ENTRADA/SALIDA DEL ADAPTADOR DE LA IMPRESORA PARALELA	
	MONITOR MONOCROMATICO	MONITOR COLOR
ENVIA DATOS (OUTPUT DATA)	3BC H	378 H
CONTROLA SALIDA (OUTPUT CONTROL)	3BE H	37A H
ESTATUTOS DE ENTRADA (INPUT STATUS)	3BD H	379 H

La figura 6.2 muestra cómo controlar las lineas de salida que van a la impresora. Estas lineas contienen todas las señales de control de salida del puerto de impresora paralela. El dato que es sacado a la impresora es enviado a un puerto diferente. Cuando se están sacando datos por este puerto, el bit 3 siempre debe estar en 1. El bit 1 debe estar en 0 para inhibir a la impresora la generación de alimentación de lineas automáticamente. Por supuesto, si se desea la alimentación de lineas automáticamente, el bit 1 debe estar en 1.

El bit 4 puede ser usado para habilitar las interrupciones de la impresora. Si el bit 4 está en 1, entonces cuando el adaptador recibe la señal de reconocimiento "acknowledge" de la impresora, se generará la interrupción IRQ7. Esta int. puede ser usada para decirle al programa (o usuario) que la impresora está lista para aceptar otro caracter. Usualmente, la señal de ocupado ("busy") es usada para determinar cuando la impresora es capaz de aceptar más datos. Este bit normalmente es 0.

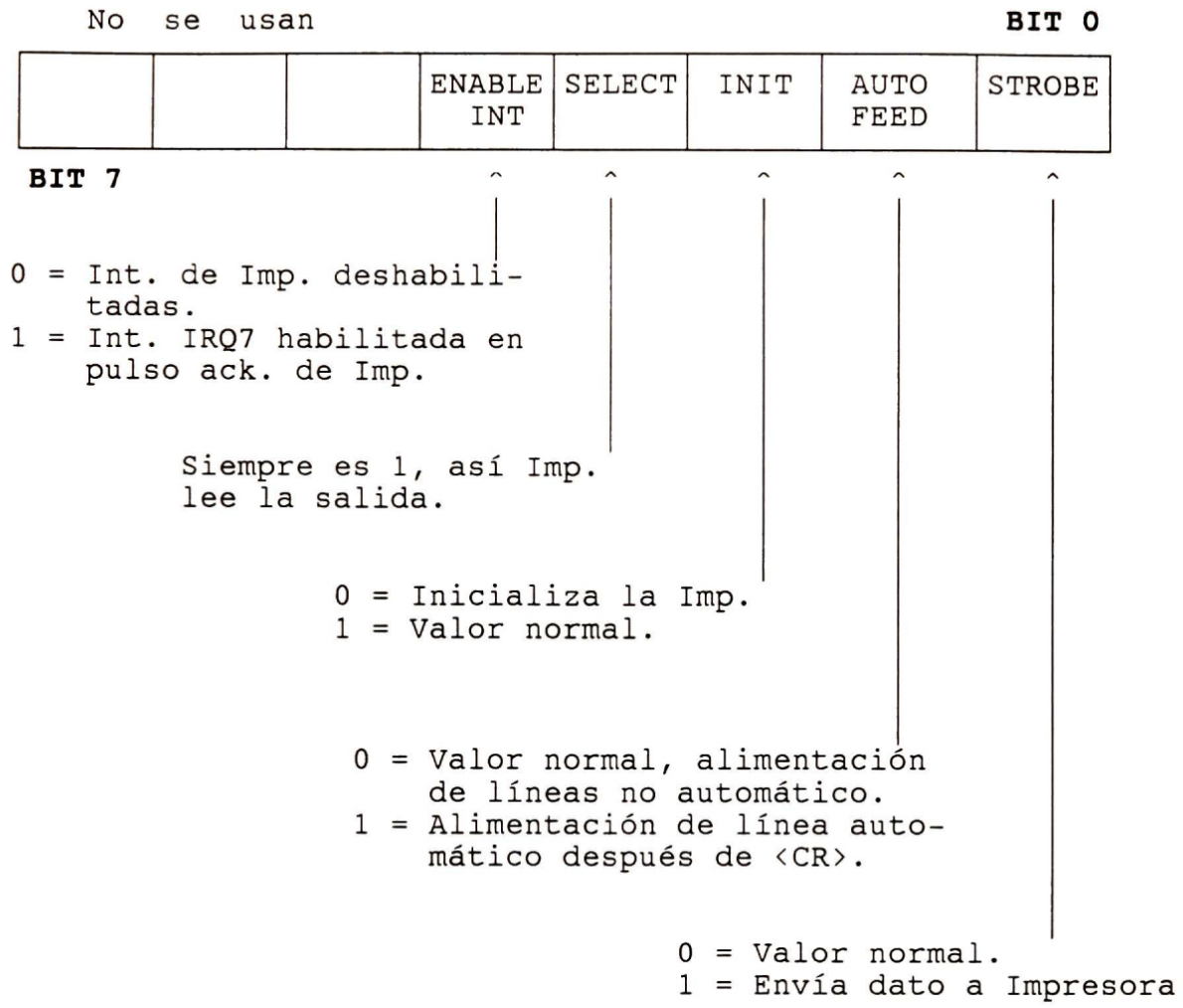


Figura 6.2.- Puerto de Control de Salida del Adaptador de la Impresora.

Cuando la impresora es inicializada, el bit 2 es puesto a 0 por lo menos 50 micro segundos. Después este período regresa a 1. Para inicializar la impresora, sacaremos un 08 H a este puerto. Después esperamos lo suficiente, sacamos un 0C h a este puerto. Finalmente la impresora ya fué inicializada.

Cuando enviamos un dato a la impresora, el bit "strobe" debe ser puesto a 1 por un período de tiempo muy corto. Este entonces debe ser reseteado a 0. El período de tiempo es así de corto que el bit puede ser puesto e inmediatamente resetearlo. Esta acción causa que la impresora lea el caracter sobre la línea de datos de la interfaz. Esto debe ser hecho solamente cuando la impresora no esté ocupada.

La figura 6.3 muestra cómo leer el estado de la impresora. Antes el dato debe ser enviado ("strobed") a la impresora, el bit de "busy" debe estar en 1.

6.3.- Manejador de Impresora.

6.3.1.- Cómo Interctúa un Proceso con el Manejador de la Impresora.

Cuando se inicializa XINIX, aparte de los procesos que se crean, tales como, disk_operator(), main_fs() y otros; también se crea el proceso que controla la impresora, este proceso se llama printer_operator().

Además de la creación de printer_operator(), previamente se asignaron (a la variable prbp, en xinix.c) 10 bloques de memoria del tamaño de la estructura ptr_req (se describe en el archivo xrpntr.h). La estructura ptr_req lleva el control de las peticiones de la impresora, a continuación se define:

```
struct ptr_req  { /* Nodo en la lista de solicitudes a la
                     impresora */
   int prpid; /* Ident. del proceso que hace la solicitud*
   unsigned int prsizet; /* tamaño de la transferencia */
   char *prbuff; /* dirección del buffer para escribir */
   char prop; /* tipo de operación: reservado para uso
                 futuro */
   int prstat; /* status que regresa OK/SYSERR */
   int prstdio; /* disp. para enviar mensaje u obtener op-
                   ciones */
   struct ptr_req *prtnext; /*Apunt. al sig. nodo en la
                               lista ptr_req */
};
```

Después de crearse el proceso printer_operator(), se verifica si (xinit(PRINTER)) PRINTER es parte de la tabla de dispositivos (en xconf.h), de ser afirmativo continúa.

El proceso printer_operator() permanece suspendido,

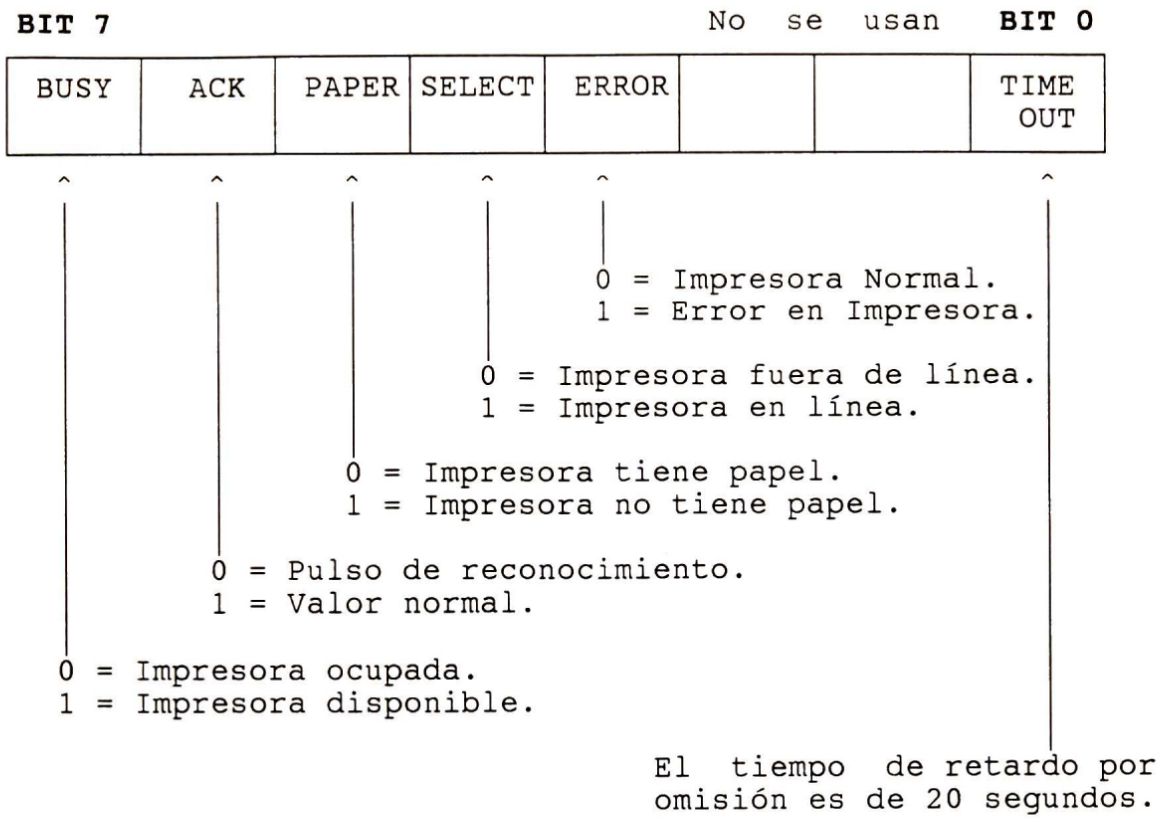


Figura 6.3.- Puerto de Control de Entrada del Adaptador de la Impresora.

mientras no exista petición a la impresora. Cuando arriva una petición, se llama a do_print(). En do_print(), inicializamos la impresora por medio de print_init().

El proceso de inicialización consiste de:

a) La obtención del contenido de la dirección absoluta de memoria 0040:0008; de donde se extrae la dirección del puerto de la impresora [15].
   - 0x378 cuando se tiene monitor de color.
   - 0x3BC cuando se tiene monitor monocromático
b) Se envía al puerto de control (0x3BC + 2) el bit 3 activo (0x08 = INIT_PRINTER, en printer.c) y el bit 2 en 0, es decir, se activa el bit de SELECT y el bit de INIT se pone en 0.
c) Se envía al puerto de control (0x3BC + 2) los bits 2 y 3, que corresponden a las señales INIT y SELECT, respectivamente.
d) Termina la inicialización de la impresora.

Una vez que se inicializó la impresora, continuamos con el desarrollo de do_print(). La tarea siguiente consiste en leer del puerto de entrada "status". Si el "status" es normal, entonces se llama a la función pr_char(); las funciones de pr_char() son las siguientes:

a) Obtiene el número de caracteres que se van a imprimir, del campo prsizet (de la estructura ptr_req).
b) Se utiliza un apuntador al campo prbuff (donde está la información que el usuario desea imprimir).
c) Lee del puerto de entrada el "status", si es normal continúa. De lo contrario, se produce un pequeño retardo y espera a que ocurra la interrupción.
d) Si el "status" es normal se envía un caracter al puerto de datos.
e) Envía al puerto de control (strobe en 1 e inmediatamente después, strobe en 0) dos bytes continuos, indicando a la impresora la llegada de un dato.
f) Se decrementa en uno la cuenta del total de caracteres que se tenía inicialmente.
g) Se produce un pequeño retardo entre la transmisión de un caracter y otro.
h) Se repite el ciclo en el paso 'c'.
j) Cuando la cuenta total de caracteres sea igual a 0, entonces se sale del ciclo y se regresa OK. También se puede abandonar el ciclo si se presenta alguna condición de error u ocupado.

En el caso de que el "status" sea diferente al normal, entonces, se pueden presentar las siguientes condiciones de error (manejado por la función pr_error()):

1) La impresora no tiene papel.
2) La impresora no está en linea.
3) Algún otro tipo de problema relacionado con los anteriores u otra condición de error diferente.

6.4.- El Comando lpr.

El comando **lpr** es utilizado por el usuario de XINIX para imprimir sus archivos, la forma de hacerlo es la siguiente:

```
CONSOLE>> lpr nombre_archivo
```

La función line_printer() (en lpr.c) lleva el control del comando lpr. Primero abre el archivo para lectura y después por medio de la función copy_to_print(), se envía el archivo a impresión. Copy_to_print() utiliza la primitiva **xread** de XINIX para leer el archivo en bloques de 1024 bytes. Después de leer un bloque lo mantiene en un buffer y posteriormente se utiliza otra primitiva de XINIX: **xwrite.**

Xread y xwrite son parte de algunos de los servicios que ofrece XINIX para el manejo del sistema de archivos. Estos servicios son de la misma naturaleza que los usados para los dispositivos (terminales, discos, impresora, etc.). Todos los dispositivos tienen un nombre mnemónico, y asociado con este un número que los identifica. XINIX maneja a los archivos como si fueran dispositivos, en el archivo xconf.h se encuentra lo siguiente:

```
#define CONSOLE     0   /* tty */
#define OTHER_1     1   /* tty */
#define OTHER_2     2   /* tty */
#define DISK0       3   /* floppy A */
#define DISK1       4   /* floppy B */
#define DISK2       5   /* disco duro */
#define PRINTER     6   /* impresora */
#define PRIM_ARCH   7   /* primer archivo */
```

Por medio de la Tabla devtab[] (ver sección 3.3.2) podemos accesar las funciones según la identificación del archivo o dispositivo. Cada entrada en devtab[] tiene la dirección de las funciones reales que ejecutan los servicios específicos como xread o xwrite.

Por ejemplo, el llamado xwrite(PRINTER, buffer, 1024) se traduce a un llamado a la función prwrite(PRINTER, buffer, 1024) del manejador de la impresora (en printer.c); y xread(ident_archivo, buffer, 1024) se traduce a un llamado a la función lfread(ident_archivo, buffer, 1024) del sistema de archivo.

6.5.- El Comando prt.

El comando **prt** también imprime archivos, con la ventaja de que el archivo se formatea en páginas, esto significa que en cada página lleva cierto formato con respecto al número de lineas y los márgenes superior e inferior.

El usuario de XINIX tiene varias opciones de impresión, veamos cómo se emplea el comado prt:

```
                  1          2              3          4    5
CONSOLE>> prt [+page] [*h header] [*l <length>] [*n] [*t]

                          file_name
```

Si utilizamos la opción 1, el comando se construye así:

```
CONSOLE>> prt +3 file_name
```

Significa que se va a imprimir el archvio file_name a partir de la página 3.

Utilizando la opción 2, tenemos que *h coloca el encabezado "header" en cada página. Si no se utiliza esta opción (*h), entonces en cada página se pone el nombre del archvio como encabezado. Además de colocar el encabezado en la parte superior izquierda de cada página del archivo, también se imprime la hora y fecha, así como el número de página, este último en el extremo superior derecho de cada página. Un ejemplo de cómo utilizar la opción 2 se muestra a continuación:

```
CONSOLE>> prt *h programa1 xinix.c
```

Tal como se mencionó en el párrafo anterior, el nombre programa1 aparecerá como encabezado en cada página del archivo xinix.c

La opción 3 limita el número de renglones por página, por "default" se utiliza 66 renglones por página. Ejemplo:

```
CONSOLE>> prt *l 55
```

En este caso particular se limita el número de renglones por página a 55.

Para el caso 4, el comando se escribe de esta manera:

```
CONSOLE>> prt *n printer.c
```

La opción *n numera cada uno de los renglones del archivo printer.c

Y por último para el caso 5:

```
CONSOLE>> prt *t xtty.h
```

Esta opción *t elimina el encabezado, es decir, el archivo se imprime por páginas pero sin encabezado.

Y si el usuario utiliza el comando sin opción alguna, entonces aparecerá como encabezado el nombre del archivo, y además de la fecha, hora y número de página. Esto es así como se indica en la siguiente página:

```
CONSOLE>> prt xconf.c
```

El encabezado se imprime en negritas (enfatizado on), se utilizan algunos caracteres de control para la impresora.

Cabe mencionar que el comando prt utiliza la interrupción 17H del BIOS, y se emplean todos los servicios de dicha interrupción (inicialización de la impresora, el obtener el "status", y el envío de caracteres).

Referencias en Bibliografía:

[13] Foerster, 1990.
[15] Young, 1988.

APENDICE 1

Usando DEBUG para ejecutar el formateo a nivel bajo.

El proceso de formateo a nivel bajo es llevado a cabo por el controlador del disco duro. Las instrucciones para operar el controlador de disco duro (CDD) están normalmente almacenadas en el ROM de esta misma tarjeta. La dirección de la memoria ROM está localizada en el banco C de la memoria de MS-DOS.

Para accesar las instrucciones almacenadas en el ROM del CDD, se utiliza el comando DEBUG de MS-DOS. Para usar DEBUG, únicamente colocarse en el subdirectorio donde se localiza este comando y teclear:

c> **DEBUG**

enseguida se carga debug y aparece el "prompt" de debug (-). Si se desea ver el contenido de la memoria ROM, la cual está localizada en la dirección 0x800 en el banco de memoria C, teclee la siguiente instrucción:

- **dc800:0**

Aparece la mitad de la pantalla con la información a partir de la dirección dada. Sobre el lado derecho de la pantalla se da la descripción del nombre del fabricante del CDD.

Desde DEBUG, podemos ordenarle al CDD que efectúe el formateo a nivel bajo. Las instrucciones para hacer esto, está localizada con un offset de 5 desde la dirección c800. De tal manera que para decirle al CDD que ejecute (GO) la orden se procede a hacer lo siguiente:

- **g=c800:5**

Al introducir este comando se produce uno de estos resultados. El CDD puede continuar y reformatear el disco duro. O este puede exhibir un menú, ofreciendo una variedad de opciones. La acción que toma el CDD depende del fabricante. Actualmente puede existir una tercera opción, es decir, el CDD no ejecuta acción alguna. Esto solamente ocurre si se tiene un CDD no estándar, el cual no coincide con los asignamientos de memoria normales de DOS.

En el caso de que el CDD muestre el menú y solicite datos de entrada tal como, el número de cabezas, número de platos, factor de interleave, etc., entonces, lo más recomendable es hacer referencia al manual de disco duro [11].

Referencia en Bibliografía:

[11] Gookin & Townsend, 1987

APENDICE 2

El comando FORMAT de MS-DOS deja preparado el disco duro o disco flexible para ser usado por MS-DOS [7].

La forma de usarlo es la siguiente:

[drive:][path] FORMAT [D:][/S][/V][/1][/4][/8]

/S -> Cuando se utiliza con ésta bandera, FORMAT coloca los archivos de sistema de MS-DOS en el disco. De esta forma se hace "booteable" el disco.

/V -> Sirve para poner un nombre hasta de 11 caracteres en el disco.

/1 -> Formatea únicamente una cara del disco.

/4 -> Formatea un diskette a doble densidad, en un drive de cuádruple densidad. Aqui el usuario debe tener precaución, porque el diskette formateado en un drive de alta densiad (cuádruple), posiblemente no se pueda leer en un drive de doble densidad

/8 -> Formatea el diskette a 8 sectores por track. Normalmente el diskette tiene 9 o 15 sectores.

Referencia en Bibliografía:

[7] Kris Jamsa, 1988

APENDICE 3

Después de teclear FDISK estando el diskette de MS-DOS en el drive "A", aparece el siguiente menú :

Fixed Disk Setup Program Version 3.30
(C)Copyright Microsoft Corp. 1987

FDISK Options

Current Fixed Disk Drive: 1

Choose one of the following:

1. **Create DOS partition**
2. **Change Active Partition**
3. **Delete DOS partition**
4. **Display Partition Information**

Enter choice: [1]

Si se desea crear una nueva partición se da 1 o se da <return> (por default ya está la opción de 1).

Supongamos que deseamos crear la nueva partición, el menú será el siguiente:

Create DOS Partition

Current Fixed Disk Drive: 1

1. **Create Primary DOS partition**
2. **Create Extended DOS partition**

Enter choice: [1]

Se selecciona "1" para crear la partición principal de MS-DOS y "2" cuando sólo es una extensión lógica de MS-DOS. Por ejemplo, cuando tenemos solamente un disco duro y lo particionamos en "C", "D" , etc.

Si deseamos cambiar la partición activa, seleccionamos la opción 2, y tendremos el siguiente menú :

```
Change Active Partition

Current Fixed Disk Drive: 1

Partition Status   Type  Start  End Size
C: 1        A   PRI DOS     0  472  473
   2            non-DOS   473  663  191


Total disk space is  664 cylinders.

Enter the number of the partition you
want to make active.............: [ ]
```

En este caso particular, unicamente podemos hacer 'booteable' la segunda partición, la cual corresponde a un sistema operativo que es diferente de DOS.

En 'Status' podemos observar la letra "A", que significa, la partición 1 es activa, o sea, 'booteable'.

El menú de arriba también señala el cilindro inicial y final de cada partición. Por ejemplo, para la primera partición tenemos que empieza en el cilindro 0 y termina en el cilindro 472.

Para borrar una partición se selecciona la opción 3 del menú principal y tendremos el siguiente enunciado :

```
Delete DOS Partition

Current Fixed Disk Drive: 1

Choose one of the following:

      1.  Delete Primary DOS partition
      2.  Delete Extended DOS partition



Enter choice: [ ]
```

Aquí, igualmente seleccionamos la partición no deseada.

Ahora, para ver la información de la tabla de particiones, se pulsa la opción 4 del menú principal y observamos lo siguiente:

Display Partition Information

Current Fixed Disk Drive: 1

Partition	Status	Type	Start	End	Size
C: 1	A	PRI DOS	0	472	473
2		non-DOS	473	663	191

Total disk space is 664 cylinders.

En conclusión es lo más importante en lo que se refiere al uso del comando FDISK. Otro punto muy importante, que debemos de considerar, es que cuando hagamos alguna modificación a la tabla de particiones, previamente debemos hacer un respaldo de toda la información del disco duro, ya que se pierde la información correspondiente a la tabla alterada [10].

Después de efectuar las particiones se procede a formatear el disco con el siguiente comando de MS-DOS:

A> **FORMAT C:\S**

Referencia en Bibliografía:

[10] Ainsbury, 1990

APENDICE 4

Descripción del FDC y del DMA

A continuación se dará una explicación breve del funcionamiento del Controlador DMA (Direct Memory Access Controller), Intel-8237 y del Controlador de disco flexible (Floppy Disk Controller, Intel-8272A) [14].

Transferencia de datos DMA (DMA Controller, 8237).

Un controlador DMA toma temporalmente los buses de datos, direcciones y control desde el microprocesador y transfiere los bytes de datos directamente desde el puerto a una serie de localidades de memoria. Debido a que la transferencia de datos es manejada totalmente en hardware, ésta es mucho más rápida que si se hiciera por medio de instrucciones de un programa. Un controlador DMA puede también transferir datos desde memoria a los puertos.

A continuación se describe como trabaja el controlador DMA. En la figura A4.1 se muestra la forma en que se conecta el controlador DMA a un sistema mínimo 8086. El punto principal que debemos tener en mente es simplemente que el microprocesador y el controlador DMA comparten el uso de los buses de direcciones, datos y control.

Cuando se enciende el sistema, los switches están en la posición superior, donde los buses están conectados desde el microprocesador al sistema de memoria y periféricos. Inicializamos todos los dispositivos programables en el sistema y ejecutamos nuestro programa hasta que necesitamos, por ejemplo, leer un archivo de disco. Para leer un archivo de disco enviamos una serie de comandos al dispositivo controlador de disco hábil, diciéndole que encuentre y lea el bloque deseado de datos desde el disco. Cuando el controlador de disco tiene el primer byte de datos del bloque del disco listo, éste envía una solicitud DMA 'request' (DREQ) al controlador DMA. Si esa entrada (canal) del controlador DMA es noenmascarable, el controlador DMA enviará una señal 'hold-request' (HRQ) a la entrada del microprocesador HOLD. El microprocesador responderá a esta entrada flotando sus buses y enviando una señal 'hold-acknowledge' (HLDA) al controlador DMA. Cuando el controlador DMA recibe la señal HLDA éste envía otra señal la cual mueve los tres buses de switches a la posición DMA, desconectando el microprocesador de los buses. El controlador DMA entonces extrae sobre el bus de direcciones la dirección de memoria donde nosotros queremos el byte de datos que va al controlador de disco. Enseguida el controlador DMA envía una señal DMA-acknowledge (DACK0) al dispositivo controlador de disco para decirle que esté listo para sacar el byte. Finalmente el controlador DMA asegura ambas lineas MEMW\ e IOR\ sobre el bus de control. Asegurando MEMW\ habilitamos la memoria

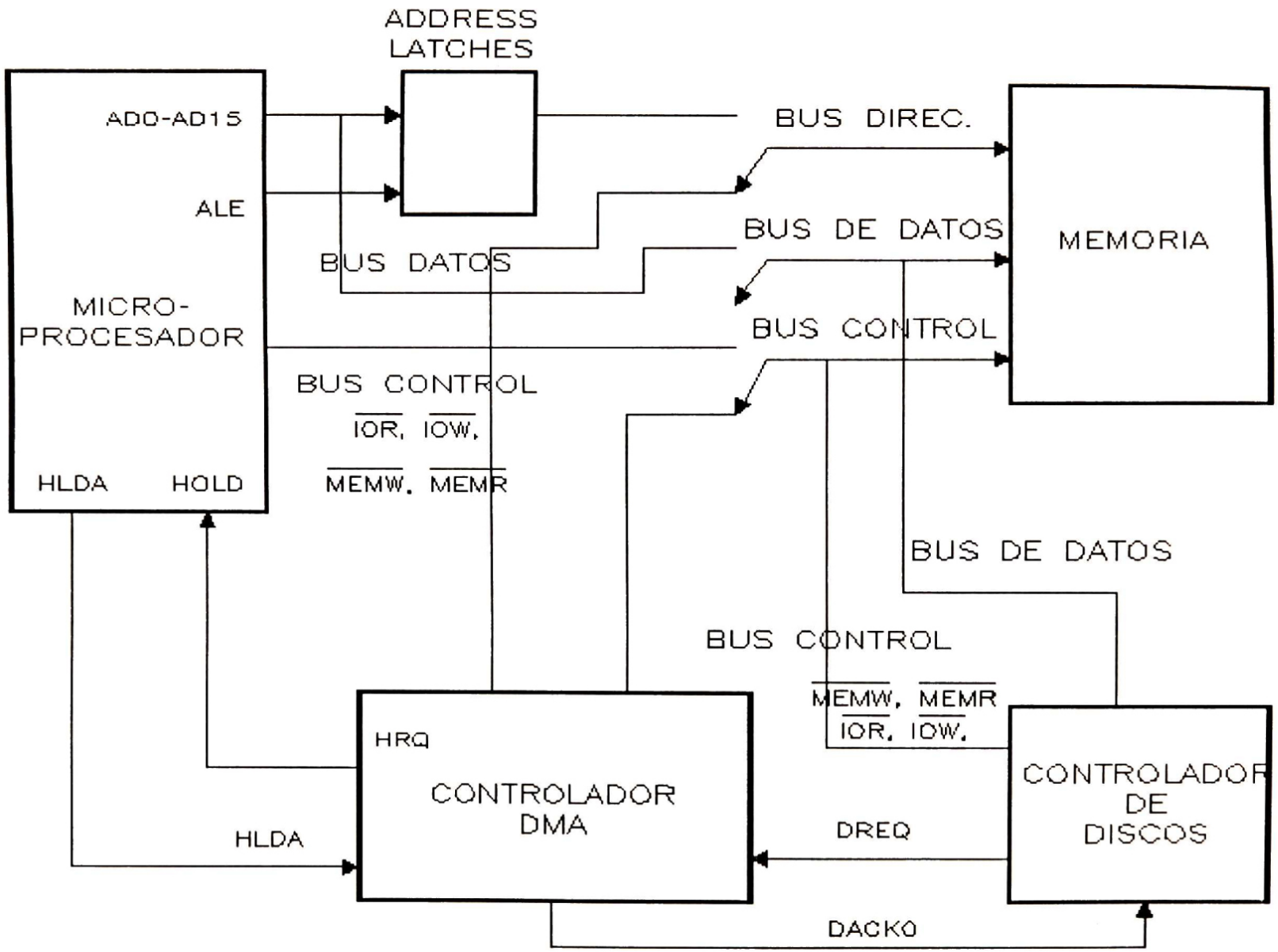

ADDRESS
LATCHES
AD0-AD15
ALE
MICRO-
PROCESADOR
HLDA
HOLD
BUS DIREC.
BUS DE DATOS
BUS DATOS
BUS CONTROL
BUS CONTROL
IOR, IOW,
MEMW, MEMR
MEMORIA
BUS DE DATOS
BUS CONTROL
MEMW, MEMR
IOR, IOW,
HRQ
CONTROLADOR
DMA
HLDA
DREQ
CONTROLADOR
DE
DISCOS
DACK0

FIGURA A4.1

direccionada para escribir datos sobre ella. Asegurando IOR\ habilitamos el controlador de disco para sacar el byte de datos desde el disco sobre el bus de datos. El byte de datos entonces se escribirá en la localidad de memoria direccioccionada.

Cuando se completa la transferencia de datos el controlador DMA suelta sus señales 'hold-request' al microprocesador, y permite que el procesador tome los buses otra vez hasta que se presente otra transferencia DMA.

Una transferencia DMA desde memoria al controlador de disco procede de manera similar excepto que el controlador DMA asegura la señal de control MEMR\ (lectura de memoria) y la señal de control IOW\ (salida - escritura). Las transferencias DMA pueden ser de sólo un byte en un tiempo o en bloques.

El controlador de disco flexible (FDC, Intel-8272A)

Escribir datos al disco flexible y leerlos después, requiere la coordinación de varios niveles. Un nivel son las señales del motor y las señales que mueven la cabeza. Otro nivel es la escritura y lectura a nivel de bit. Y por último tenemos el nivel de interfase con el resto de la circuitería del microprocesador. Llevar a cabo la coordinación de todo esto es un trabajo bastante complicado, de esta manera, se utiliza el controlador del disco flexible diseñado especialmente para hacer esto.

En la figura A4.2 se observa un diagrama a bloques del 8272A. Las lineas del bus de datos, RD\ , WR\, A0, RESET\, y CS\ son las señales de interfase de los periféricos estandares. Las señales DRQ, DACK\, e INT son usadas para la transferencia DMA de datos para y desde el controlador.

Es necesario dar una breve explicación de estas señales para tener un mejor panorama de como se relacionan con el hardware de la unidad de disco flexible.

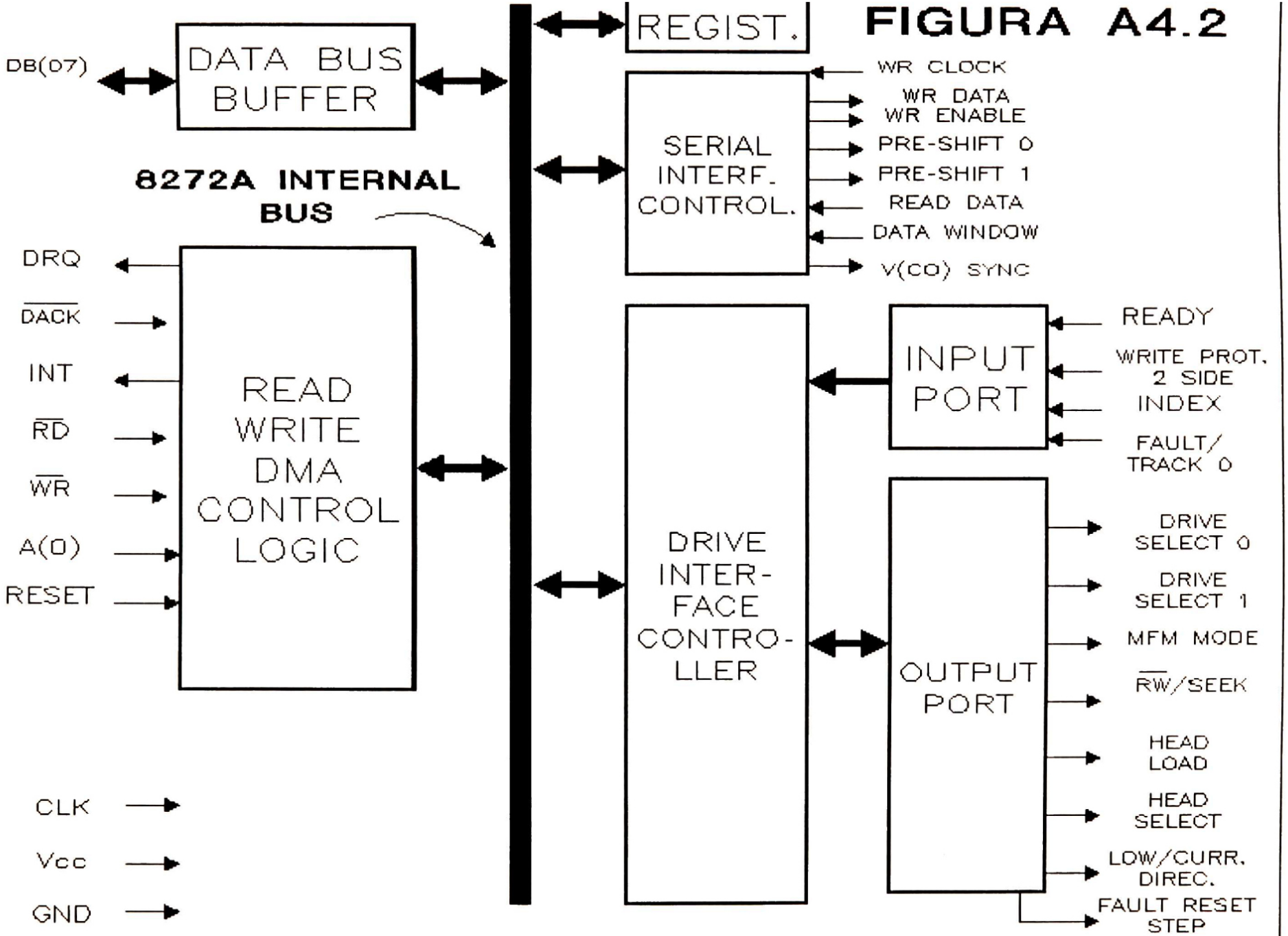
La señal de entrada **READY** de la unidad de disco debe ser alta si la unidad está encendida y lista para iniciar. Si, por ejemplo, tú olvidas cerrar la puerta de la unidad de disco, la señal READY no cambia a alto.

La señal **WRITE PROTECT/TWO SIDE** indica si la muesca o ranura de protección contra escritura está cubierta cuando el drive está en el modo de lectura o escritura. Cuando el drive está operando en el modo búsqueda de track, esta señal indica si el drive es de dos lados o un lado.

La señal **INDEX** se pulsará cuando el agujero índice del disco pasa entre el LED y el detector fototransistor.

La señal **FAULT/TRACK0** indica alguna condición de error durante una operación de lectura/escritura. Cuando se lleva a cabo una operación de búsqueda de tracks ésta señal deberá ser mantenida cuando la cabeza está sobre el track cero, el track más exterior sobre el disco.

FIGURA A4.2


DB(07)
DATA BUS BUFFER
REGIST.
8272A INTERNAL BUS
SERIAL INTERF. CONTROL.
WR CLOCK
WR DATA
WR ENABLE
PRE-SHIFT 0
PRE-SHIFT 1
READ DATA
DATA WINDOW
V(CO) SYNC
DRQ
DACK
INT
RD
WR
A(0)
RESET
READ WRITE DMA CONTROL LOGIC
INPUT PORT
READY
WRITE PROT.
2 SIDE
INDEX
FAULT/
TRACK 0
DRIVE INTER-FACE CONTRO-LLER
OUTPUT PORT
DRIVE SELECT 0
DRIVE SELECT 1
MFM MODE
RW/SEEK
HEAD LOAD
HEAD SELECT
LOW/CURR. DIREC.
FAULT RESET STEP
CLK
Vcc
GND

La señal de salida **DRIVE SELECT,** DS0 Y DS1, desde el controlador son enviadas a un decodificador externo el cual usa estas señales para producir una señal que habilita una de cuatro unidades.

La señal de salida **MFM** debe mantenerse alta si el controlador está programado para modulación de frecuencia modificada (MFM), y la señal es baja si el controlador es programado para modulación de frecuencia estándar (FM).

La señal **RW\ /SEEK** es usada para decirle a la unidad que opere en modo lectura-escritura o en modo búsqueda-track.

La señal **HEAD LOAD** es mantenida por el controlador para decirle al hardware de la unidad, que coloque la cabeza de escritura/lectura en contacto con el disco. Cuando se hace la interfase a una unidad de doble densidad, la señal HEAD SELECT desde el controlador es usada junto con esta señal para indicar cual de las dos cabezas debe ser cargada.

Durante las operaciones de escritura sobre los tracks internos del disco la señal **LOW CURRENT/DIRECTION** es mantenida por el controlador.

La señal de salida **FAULT RESET/STEP** es usada para resetear la falla del flip-flop después de que sucedió una falla, corregida cuando se está efectuando un comando de escritura o lectura. Cuando el controlador está llevando un comando de búsqueda de track, este pin es usado para sacar los pulsos con los cuales la cabeza pasa desde un track a otro.

Comandos del 8272.

El C.I. 8272 puede ejecutar 15 comandos diferentes. Cada uno de estos comandos es enviado al registro de datos en el controlador como una serie de bytes.

Después de que un comando fué enviado al 8272, este envía el comando, y regresa el resultado en el registro de status del 8272, y/o al registro de datos en el 8272. En los siguientes párrafos tenemos una descripción breve de los comandos:

SPECIFY - Inicializa el tiempo de carga de la cabeza.

SENSE DRIVE STATUS - Regresa la información del status del drive.

SENSE INTERRUPT STATUS - Sensa la señal de interrupción del 8272.

SEEK - Posiciona la cabeza de lectura/escritura en el track especificado.

RECALIBRATE - Posiciona la cabeza sobre el track '0'.

FORMAT TRACK - Escribe el campo ID, gaps, y marcas de dirección sobre el track.

READ DATA - Carga la cabeza, y lee una cantidad especificada de datos desde el sector.

READ DELETED DATA - Lee datos desde los sectores marcados como eliminados.

WRITE DATA - Carga la cabeza, y escribe datos en el sector especificado.

WRITE DELETED DATA - Escribe las marcas de las direcciones de los datos eliminados en el sector.

READ TRACK - Carga la cabeza y lee todos los sectores en el track.

READ ID - Regresa el primer campo ID encontrado en el track.

SCAN EQUAL - Compara un sector de bytes de datos leídos desde disco con los bytes de datos enviados de CPU o del controlador DMA hasta cadenas iguales. Coloca el bit en el registro de status si son iguales.

SCAN HIGH OR EQUAL - Coloca la bandera si la cadena de datos del sector del disco es más grande o igual a la cadena de datos del CPU o controlador DMA.

SCAN LOW OR EQUAL - Coloca la bandera si la cadena de datos del sector del disco es menor o igual a la cadena de datos de CPU o controlador DMA.

Referencia en Bibliografía:

[14] Hall, 1986

BIBLIOGRAFIA

[1] Tesis "XINIX Sistema Operativo para Computadora Personal" de Jorge Buenabad Chávez, 1989.

[2] Tesis "Sistema de Archivos para el Sistema Operativo XINIX" de Ruth E. Delgado Moreno, 1990.

[3] Operating Systems A Pragmatic Approach. 1973
de Harry Katzan, Jr. Edit. Van Nostrand Reinhold C.

[4] Operating Systems Principles. Segunda edición 1971
de Stanley A. Kurzban, Thomas S. Heines y
Anthony P. Sayers Edit. Van Nostrand Reinhold C.

[5] Operating System Design Vol. 1: 1988
The XINU Approach (PC Edition).
de Douglas Comer y
Timothy V. Fossum Edit. Prentice Hall

[6] Sistemas Operativos: Diseño e Implementación. 1988
de Andrew S. Tanenbaum Edit. Prentice Hall

[7] DOS Power User's Guide. 1988
de Kris Jamsa Edit. McGraw-Hill

[8] MS-DOS Developer's Guide. Segunda edición 1989
de Angermeyer-Jaeger,
Kumar-Barkakati,... Edit. Howard W. Sams & Company

[9] The New Peter Norton Programmer's Guide to The IBM PC & PS/2. 1988
de Peter Norton y
Richard Wilton Edit. Microsoft Press

[10] Using Your hard disk. 1990
de Bob Ainsbury Edit. QUE

[11] Hard disk Management with MS-DOS and PC-DOS. 1987
de Gookin & Townsend Edit. TAB

[12] Introducción a los Sistemas Operativos 1987
de Harvey M. Deitel

[13] The Printer Bible. 1990
de Scott Foerster Edit. QUE

[14] Microprocessors and Interfacing
Programming and Hardware. 1986
de Douglas V. Hall Edit. McGraw-Hill

[15] Systems Programming in Turbo C. 1988
de Michael J. Young Edit. SYBEX

[16] Advanced MS-DOS Programming. Segunda edición 1988
de Ray Duncan Edit. Microsoft Press

Turbo C Bible. 1989
de Nabajyoti Barkakati Edit. Howard W. Sams & Company

Turbo C The Complete Reference. 1988
de Schildt Edit. McGraw-Hill

C Programmer's Toolkit. 1989
Jack Purdum Edit. QUE

Programming with Turbo C. 1989
de S. Scott Zimmerman y
Beverly B. Zimmerman Edit. Scott, Foresman & Company

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL I.P.N.

EL JURADO DESIGNADO POR LA SECCION DE COMPUTACION DEL DEPARTAMENTO DE INGENIERIA ELECTRICA, APROBO EL DIA 7 **DEL MES DE** MARZO **DEL AÑO DE** 1992. **EL TRABAJO DE TESIS** "SISTEMA OPERATIVO XINIX CON MANEJADOR DE DISCO DURO E IMPRESORA"

DESARROLLADO POR EL ALUMNO: DAVID LEIJA DIAZ

M EN C. JORGE BUENABAD CHAVEZ
Profesor Investigador CINVESTAV 1-B
Sección de Computación

M EN C. ANDRES VEGA GARCIA
Profesor Investigador CINVESTAV 1-B
Sección de Computación

DR. SERGIO V. CHAPA VERGARA
Profesor Investigador CINVESTAV 3-A
Sección de Computación

Ave. Instituto Politécnico Nacional 2508 Col. San Pedro Zacatenco México, D.F. C.P. 07300
Tels: 752-0677 y 754-0200 Telex: 1772826 PPTME Fax: 752-0590 y 586-6290

AUTOR LEIJA DIAZ, D.

TITULO SISTEMA OPERATIVO XINIX CON MANEJADOR DE DISCO...

CLASIF. XM 92.14 RGTRO. BI 12860

NOMBRE DEL LECTOR	FECHA PREST	FECHA DEVOL.
Dr. Guillermo Morales	11-12-95	13 DIC 95
Victor J. Ceron Lorta	10/09/02	12/sept/02
Salvador Castro Enzso	09/10/02	23/10/02
Salvador Castro Enzso	20/1/03	20/1/03