Periféricos

Javier Gil. D.F.I.S.T.S.

Curso 2005-2006

Índice general

Capítulo 1

Presentación de la asignatura

1.1. Advertencia

La asignatura de Periféricos ha evolucionado mucho a lo largo de los últimos años, tratando siempre de buscar un difícil compromiso entre teoría y práctica, y tratando también de no adherirse a implementaciones particulares, sino de proporcionar conocimientos básicos generales, aplicables en multitud de situaciones, así como descripciones físicas de las tecnologías implicadas. Por estos motivos, y en primer lugar, debería quedar claro de lo que *no* va a tratar esta asignatura: no vamos a aprender a montar ordenadores por piezas, a colocar tarjetas de ampliación o a configurar controladores software.

1.2. Un poco de historia

Aunque los periféricos como tales son un producto de la era informática, nacidos a partir de unas necesidades concretas, también es cierto que algunos de ellos tiene sus ancestros en las tecnologías que emergieron a raíz de la revolución industrial.

Los periféricos pueden ser divididos según su uso en periféricos de entrada/salida y periféricos de almacenamiento. De entre los primeros, los más obvios son los teclados, provenientes directamente de las máquinas de escribir, de las cuales conservan la disposición de las teclas, y aunque en las últimas décadas se han añadido algunas mejoras, este periférico sigue siendo esencialmente el mismo que definió IBM para sus estaciones de trabajo en los años sesenta. No obstante, debemos destacar las máquinas de oficina diseñadas por Honey-Well y la propia IBM, como sumadoras y artefactos

mecánicos para la cumplimentación de formularios, que pueden verse como implementaciones mecánicas o electro-mecánicas de un software aún muy primitivo.

Pero, volviendo a los periféricos de entrada-salida, hay que decir que en los primeros años de la era informática estos consistían casi siempre en desarrollos específicos para los ordenadores y sistemas operativos fabricados por las compañías pioneras a partir de los años cincuenta. Por ejemplo, recuérdense las perforadoras y lectoras de tarjetas.

Directamente relacionados con los teclados encontramos desde los primeros años del desarrollo de los ordenadores los dispositivos de impresión, derivados de las imprentas existentes desde el siglo XV pero que pueden asimismo verse como un desarrollo inverso al del teclado, y conceptualmente muy próximo. Así, mientras que en este último una acción mecánica provoca una señal eléctrica, en el primero es la señal eléctrica la que provoca la acción mecánica. Pronto llegaron refinamientos para hacerlas más veloces y fiables, y para incrementar la calidad del texto impreso. Las primeras impresoras matriciales aumentaron el número de elementos de la matriz de impacto, y redujeron el tamaño de las partes móviles, con el fin de disminuir las inercias. Cuando esto no fué suficiente, se desarrollaron impresoras de líneas, que multiplicaban por uno o dos órdenes de magnitud la velocidad de impresión. El paso de una informática profesional a una informática personal, con la correspondiente expansión a ambientes que habían permanecido ajenos hasta entonces, como las fábricas y oficinas, creó dos necesidades fundamentales: silencio y velocidad. Estas necesidades fueron cubiertas mediante el desarrollo de impresoras de inyección de tinta e impresoras *laser*, que no han cesado de mejorar desde entonces.

A finales de los años sesenta, la informática estaba lo suficientemente madura como para dar un paso cualitativo importante con un aumento de la interactividad entre los operadores humanos y el software. Hasta entonces, ésta se había limitado a la emisión de mensajes a través de una impresora en línea, pero en la época a la que nos referimos ya era posible usar monitores, con los que se consiguió dar una imagen electrónica del espacio de trabajo que el usuario hasta entonces había usado como un modelo conceptual de su relación con la máquina: ahora por fin era posible ver ese espacio. Durante aproximadamente una década, la tecnología no tuvo capacidad más que para mostrar imágenes textuales, con una resolución limitada. Podemos decir que del modelo abstracto en la mente del operador se había pasado a un modelo

basado en palabras, cuya representación física se manifestaba a través de un monitor.

Pero a finales de los años setenta, en los laboratorios de Xerox se diseño una nueva forma de enfocar las relaciones entre usuarios y máquinas. El nuevo camino abierto sustituía la representación textual por una representación simbólica basada en iconos. Consecuentemente, hubo que sustituir el dispositivo de entrada básico, el teclado, por un nuevo periférico que se adaptara de forma más natural a la concepción que acababa de nacer. De esta forma se diseñó el popular *ratón*. Desgraciadamente, esta representación pictográfica es conceptualmente inferior a la representación textual, a la que ha sustituido casi completamente, impidiendo de paso el que la primera se desarrolle hacia una mejor usabilidad. El motivo es que la representación textual puede usarse como interfaz universal entre el usuario y los programas y entre programas distintos, mientras que no existe una interfaz general que permite conectar iconos.

El diseño original del sistema operativo UNIX tuvo por lo que respecta a esta exposición dos aspectos destacables. Por una parte, el diseño contempló desde un primer momento la interconexión de ordenadores, formando redes. Esto no sólo implica una notable visión de futuro, año 1969, sino que indica un estado tecnológico que permitía producir puertos de comunicaciones y sistemas de modulación-demodulación. El segundo aspecto destacable es la concepción de los periféricos desde un nivel conceptual superior como archivos del sistema, donde era posible leer y escribir una corriente de octetos. La información que fluye a y desde estos archivos carece para el sistema operativo de todo significado: son los programas de aplicación los que interpretan los datos según sus necesidades.

Sin duda, son los sistemas de almacenamiento masivo los que han tenido una influencia más determinante en el desarrollo de la informática de las últimas décadas. Los origenes los encontramos en el computador *ABC* (Athanasoff-Berry Computer) construido entre los años 1937-42. Aunque no fué el primer computador digital, fue el primero en donde los elementos de cómputo estaban separados de los elementos de memoria. Estos se realizaban mediante condensadores montados en grandes discos de baquelita. A este antecedente temprano siguieron las memorias de núcleo de ferrita, y en ambos casos eran precisos circuitos de refresco que evitaran la pérdida de los datos.

Por eso tiene gran importancia el desarrollo por IBM en los años cincuenta de memorias permanentes realizadas sobre un material ferromagnético al que se somete localmente a un campo magnético superior al campo coercitivo del mismo. Este material ferromagnético se deposita sobre una superficie no magnética, como el vidrio o la baquelita. Nació de esta forma el disco duro, y con él se multiplicó por varios órdenes de magnitud la cantidad de información que podía almacenarse de forma permanente. Durante dos décadas, este fué un periférico costoso con una utilidad limitada a aplicaciones concretas. Por este motivo, en el diseño del ordenador personal no estaba previsto un disco duro, aunque sí una o dos unidades de discos flexibles, con capacidades de 160 kilobytes. Esta fué una estimación excesivamente conservadora, aunque este juicio es ahora fácil de hacer. En realidad, no había razón para pensar que discos de ese tamaño no podían satisfacer las necesidades de almacenamiento normales. Lo que sucedió fué que la posibilidad de almacenar y compartir información creo la necesidad de hacerlo, y con ella la necesidad del software necesario para producirla y gestionarla, que a su vez debía ser almacenado en algún sitio. Pronto se añadió un disco duro de 10 Megaoctetos, que creció enseguida a 20 Megaoctetos. En los veinte últimos años, esta cantidad ha crecido en más de dos órdenes de magnitud, y no parece que la tendencia remita. Para satisfacerla, se han desarrollado interfaces adecuadas para acceder a los discos rígidos de forma rápida, memorias intermedias para acelerar aún más el proceso, unidades de disco flexible más precisas, capaces de alcanzar los centenares de Megaoctetos de capacidad, y, en fin, otros métodos para almacenamiento masivo basados en fenómenos magneto-ópticos u ópticos. Al mismo tiempo, se ha mejorado sustancialmente la velocidad de operación de los medios de almacenamiento secuenciales tradicionales, y se vislumbran ya capacidades superiores de almacenamiento, y nuevos y prometedores métodos, como las memorias holográficas.

Por otro lado, la potencia creciente de computación asociada a cada nueva generación de procesadores ha permitido la exploración práctica del nuevo mundo del reconocimiento del habla y la realidad virtual. La primera, puede convertir en obsoletos los sistemas de entrada tradicionales. La comunicación mediante el habla permitirá una interacción más fácil y rápida, pero a este reto han de hacer frente aún las nuevas generaciones de sistemas operativos, aunque OS/2 dió el primer paso en esa dirección, al incorporar reconocimiento del habla de forma nativa. Otros sistemas, como BeOS, han sido diseñados específicamente pensando en las ingentes cantidades de información que habrán de ser almacenadas y procesadas.

Esta potencia de cálculo puede usarse asimismo para explorar una nueva representación del espacio conceptual que usa el ser humano para comunicarse con la máquina, y representar las operaciones que hace sobre ella en un espacio tridimensional virtual. Pero para que esto suceda, es preciso el advenimiento de una nueva generación de periféricos, como guantes táctiles y cascos de visión estereoscópica con sonido integrado.

Los periféricos juegan de esta forma un papel determinante en el desarrollo del software, y en concreto de los sistemas operativos. También en el desarrollo de las nuevas generaciones de procesadores. Ya que, por una parte, estimularán el desarrollo de los primeros [1], que a su vez, al acomodar de forma natural nuevos periféricos traerán consigo mejoras y refinamientos en estos últimos. En cuanto a los procesadores, es obvio que habrán de dar respuesta a los nuevos retos, incrementando nuevamente su potencia. Y no olvidemos que, a otro nivel, el de los programas de aplicación, también se producirá una revolución en la forma de concebirlos [2].

1.3. Dos perspectivas

Antes de entrar en materia, conviene tomar cierta distancia, y reflexionar sobre los periféricos en general. Una forma de hacerlo es indagando sobre las relaciones entre tres elementos: La potencia de cálculo disponible en los sistemas informáticos, el diseño de los Sistemas Operativos, y los Periféricos. Pueden identificarse relaciones estrechas entre cualquier par de ellos. Centrémonos para empezar en la pareja Sistema Operativo - Periféricos. El diseño de los primeros está determinado por las características de los segundos. En concreto, con su escasa velocidad en comparación con la velocidad de cálculo y en el hecho de que los periféricos funcionan de forma asincrónica con el sistema. Es decir, el Sistema no puede prever que, un milisegundo después, un periférico requerirá su atención. En los casos extremos, la necesidad de atender a un periférico en un tiempo acotado y pequeño puede tener un impacto determinante en el diseño de todo el Sistema Operativo. Por ejemplo, en el Sistema Operativo Solaris el núcleo puede ser expropiado, garantizando de esta forma que ciertos periféricos serán atendidos inmediatamente. En sentido inverso, la influencia es también determinante. Los periféricos tienen una parte física y sobre esa parte física se interacciona mediante unos protocolos convenidos. El Sistema Operativo debe por supuestos adaptarse a estos

[1] véase por ejemplo el conjunto de instrucciones MMX que introdujo Intel

[2] Siempre que se produzca un cambio igual de radical en la forma y las herramientas para producir programas

protocolos, de forma que los programas de aplicación puedan sacar provecho en su relación con los periféricos. Otras veces, el Sistema Operativo provee pautas para el diseño de los periféricos. Por ejemplo, los periféricos *Plug and Play* deben ser capaces de ofrecer al Sistema Operativo las informaciones que éste necesite.

Reflexionemos ahora sobre la pareja Potencia de cálculo - Periféricos. Es usual que la potencia de cálculo disponible no sea suficiente para acomodar de forma adecuada ciertos periféricos. Entonces, se tiende a delegar en el propio periférico parte de la potencia de cálculo. Existen muchos ejemplos: desde la gestión del disco que realiza una controladora IDE, pasando por la cierta *inteligencia* que se deposita en un subsistema SCSI hasta las tarjetas gráficas sofisticadas con potencias de cálculo totalmente equiparables a la de la propia UCP. En sentido inverso, también pueden encontrase relaciones. Por ejemplo, el control del ordenador por voz no fué posible hasta disponer de potencia de cálculo suficiente.

Visto desde el punto de visto cronológico, podemos identificar cuatro fases en el desarrollo de los periféricos. En la primera fase, no existe realmente diferencia entre el periférico y el instrumento de cómputo. Esta es la época de los artefactos mecánicos y electromecánicos hasta cierto punto programables.

En una segunda fase, ya con el modelo de von Neumann implementado, se produjo la separación entre los periféricos y los dispositivos de cálculo. En esta fase, los periféricos son aún casi totalmente pasivos.

En la tercera fase, la potencia de cálculo comienza a distribuirse entre los periféricos: controladores de disco inteligentes, gestores para los buses, aceleradoras gráficas que gestionan decenas de Megabytes de RAM, por poner unos ejemplos.

Finalmente, se alcanza la cuarta fase cuando, de nuevo, se produce una convergencia entre los dispositivos de cálculo y los periféricos. Nos encontramos ahora en los albores de esta fase, con el diseño y primeras implementaciones de los sistemas operativos distribuidos, el almacenamiento distribuido de la información, la migración transparente de procesos entre máquinas y, en general, con la compartición de recursos a través de las redes y la gestión distribuida de esos recursos.

1.4. Plan del curso

Teniendo en mente la intención de ofrecer conocimientos generales basados en modelos o descripciones de la tecnología, y evitar en lo posible implementaciones particulares, el curso tendrá cuatro bloques bien diferenciados.

En el primero de ellos, se estudiará el problema general de la entrada/salida, las estrategias a las que recurren los sistemas operativos para dar respuesta eficiente a las necesidades específicas de los periféricos (atención asíncrona y lenta velocidad de proceso) y una visión general de los interfaces de conexión.

El segundo bloque se referirá a la conexión con el *mundo exterior* a través de los puertos de comunicaciones e interfaces de red. Dentro de este bloque tiene cabida una exposición de las tecnologías de impresión, con un capítulo dedicado al lenguaje de descripción de páginas PostScript, así como una exposición de la base de datos *termcap* y la integración de terminales en sistemas tipo *NIX.

El tercer bloque es el de mayor peso específico, y en él se abordará el problema del almacenamiento masivo y permanente de la información. Se describirán los fundamentos físicos, diseño y gestión general de los dispositivos de almacenamiento magnético y se presentarán otras variantes, como almacenamiento óptico, magneto-óptico, holográfico y orgánico. Asímismo se expondrán algunos modelos sencillos que ayuden a comprender la gestión de estos medios por parte de los sistemas operativos.

En el cuarto bloque, se considerarán los dispositivos de visualización, haciendo un repaso de las tecnologías empleadas y exponiendo algunos problemas de gestión específicos.

Capítulo 2

Consideraciones generales sobre la entrada/salida

2.1. El problema general de la entrada/salida

Existe una cantidad considerable de dispositivos asociados a los sistemas informáticos, y cada uno tiene sus propias especificaciones físicas y lógicas, y opera a una velocidad característica. Es por esto que los sistemas operativos han de tener un subsistema de E/S encargado de atender a todos estos dispositivos, ocultando a las capas superiores las particularidades de cada uno.

Por otra parte, el hardware puede controlar de forma autónoma muchas de las operaciones de E/S, y es una tendencia creciente la incorporación a los sistemas informáticos de procesadores especializados para determinadas operaciones, convirtiéndose en verdaderos sistemas multiprocesador. La mayoría de los dispositivos de E/S comparten dos características:

- operación asíncrona
- diferencia de velocidad con la UCP

2.1.1. Operación asíncrona

Cuando un programa se ejecuta, el procesador está realizando sucesivamente comunicación con la memoria principal y con los dispositivos de E/S. Sin embargo, el acceso a memoria se produce en un tiempo casi constante, es

decir, la memoria está siempre dispuesta para entregar datos al procesador [1] . Como este tiempo es pequeño, el procesador no hace nada hasta que recibe los datos solicitados.

Pero la comunicación con el mundo exterior es distinta, pues no tiene relación alguna con el reloj del procesador: la llegada de datos es impredecible. Esto se llama 'operación asíncrona', y obliga al hardware de E/S a implementar señales de control especiales que capacitan al procesador y a los controladores de dispositivos para que se indiquen mútuamente sus estados, intenciones y disponibilidad de datos. Cuando el procesador quiere acceder a un dispositivo de E/S comprueba su estado. Si está disponible, se efectúa la petición, si no, el S.O. puede esperar o puede dar el control a otro proceso.

2.1.2. Diferencia de velocidad

La velocidad del bus entre procesador y memoria es del orden de millones de octetos por segundo, mientras que la velocidad de los dispositivos de E/S varían entre 1 octeto por segundo y varios millones de octetos por segundo. Los primeros son impresoras matriciales y terminales lentos, mientras que los segundos pueden ser discos de alta velocidad comunicados por enlaces ópticos. Usualmente, estos últimos tienen procesadores dedicados y suelen operar mediante DMA.

Véanse por ejemplo las velocidades de transferencia de los periféricos del sistema Sun Enterprise 6000 (valores aproximados y relativos):

[1] Esto era realmente así hasta principios de la década de los 90. En algún momento en esta época se produce el cruce entre la curva que representa la velocidad de la memoria a lo largo del tiempo y la curva que representa la velocidad del procesador. Desde entonces, la velocidad del procesador ha crecido proporcionalmente mucho más.

teclado	10^{-2}
raton	$5 * 10^{-2}$
modem	100
impresora laser	500
ethernet	1000
disco duro	8000
ethernet rapida	10000
bus SCSI	40000
SBUS	100000
bus gigaplano	1000000

Consideremos un terminal a 2000 cps, lo que supone un carácter cada $5 * 10^{-4}$ segundos. Puesto que el procesador puede ejecutar una instrucción en $0{,}1{*}10^{-6}$ segundos, significa que, si estuviese dedicado a E/S, permanecería inactivo más del 99 % del tiempo.

Una manera de mejorar la utilización del procesador es multiplexando entre varios trabajos, lo que requiere asistencia hardware en forma de *buffers* de datos, señales de diálogo y mecanismos de sincronización.

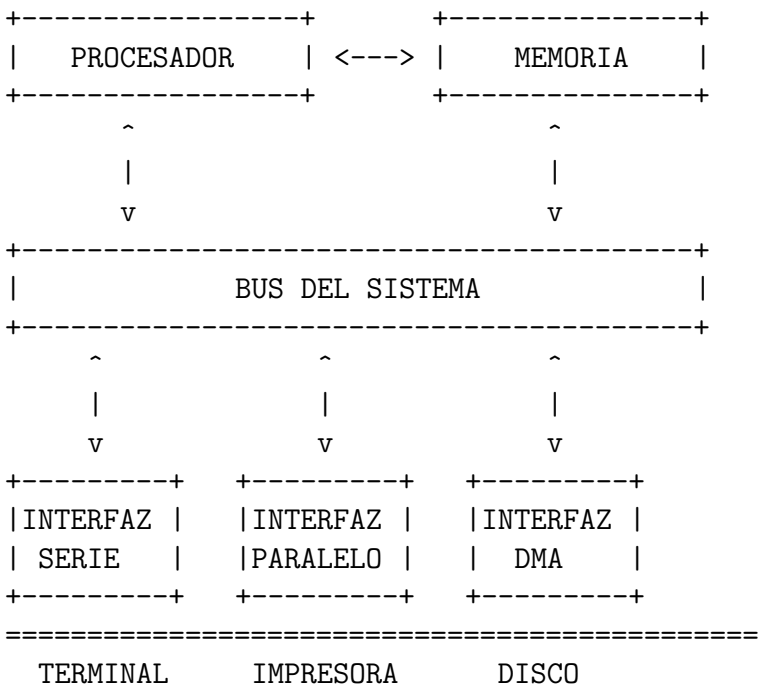
2.2. Interfaces de E/S

Los controladores hardware actúan como intermediarios entre los dispositivos E/S y el sistema. Estos interfaces tienen dos misiones principales: traducir las señales genéricas del procesador a las señales específicas del dispositivo y armonizar de alguna forma la diferencia de velocidad.

El hardware de E/S esta distribuido sobre una serie de interfaces, conectadas a la placa base. A su vez, los conectores están conectados al bus del sistema, que sirve como medio de comunicación para intercambios de direcciones, datos y señales de control.

Visto con algo más de detalle, la arquitectura PC basada en bus PCI consta funcionalmente de los siguientes elementos:

- El procesador: éste se comunica directamente con su caché interna, y con la memoria principal a través de un controlador de memoria que a su vez está conectado al bus PCI.

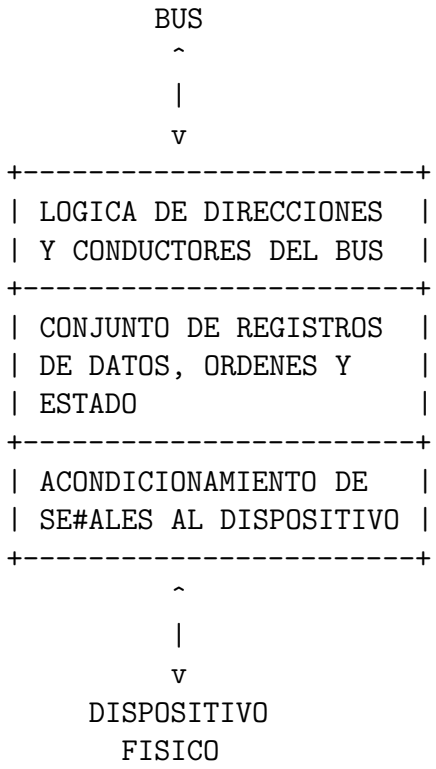


- El controlador de gráficos: por una parte, conectado al bus, y por otra al monitor, a quien envía la señal que forma la imagen.
- Un posible controlador SCSI: está conectado con el bus PCI por un lado y con un bus independiente SCSI, compartido por los distintos dispositivos SCSI.
- Un controlador de disco IDE: conectado con el bus del sistema y con hasta 4 dispositivos IDE.
- Un controlador de un bus de expansión, compartido por teclado y puertos serie y paralelo.

Una tarjeta de interfaz está diseñada para gestionar uno o más dispositivos del mismo tipo. Los distintos dispositivos pueden conectarse a la tarjeta mediante canales dedicados o en cascada. El primer método suele usarse en terminales, y el segundo en unidades de disco. Un controlador de E/S está dividido en tres capas funcionales:

1. Interfaz del bus
2. Controlador genérico de dispositivo

3. Interfaz del dispositivo



En el extremo del bus, cada controlador debe aparecer como un módulo estandard capaz de gestionar lecturas, escrituras, arbitrajes de prioridad, interrupciones y ciclos DMA. En el extremo del dispositivo, el controlador debe proporcionar las señales eléctricas que necesita el dispositivo.

Las dos capas extremas son de interés para los diseñadores hardware. La intermedia, para el diseñador software, ya que proporciona a los programadores de sistemas una abstracción uniforme de las operaciones de E/S. Normalmente, cada dispositivo aparece como un conjunto de registros dedicados, llamados *puertos de E/S*. Por ejemplo, tenemos puertos serie, paralelo, controladores de discos flexibles y rígidos, controladores gráficos, temporizadores y controladores de red. En general, los puertos son bidireccionales, y por convenio se suele adoptar el punto de vista del procesador cuando hablamos de 'entrada' y 'salida'. Los puertos tienen registos de datos, registros de órdenes y registros de estado.

Aunque a los puertos puede accederse a través de un espacio propio de direcciones, en otros casos el acceso se realiza a través de posiciones específicas de RAM. Cada método tiene ventajas e inconvenientes.

La siguiente tabla muestra algunas de las direcciones de puerto en la arquitectura PC:

000-00F	controlador DMA
020-021	controlador de interrupciones
040-043	temporizador
200-20F	controlador de juegos
2F8-2FF	puerto serie secundario
320-32F	controlador de disco duro
378-37F	puerto paralelo
3D0-3DF	controlador de gráficos
3F0-3F7	controlador de disco flexible
3F8-3FF	puerto serie primario

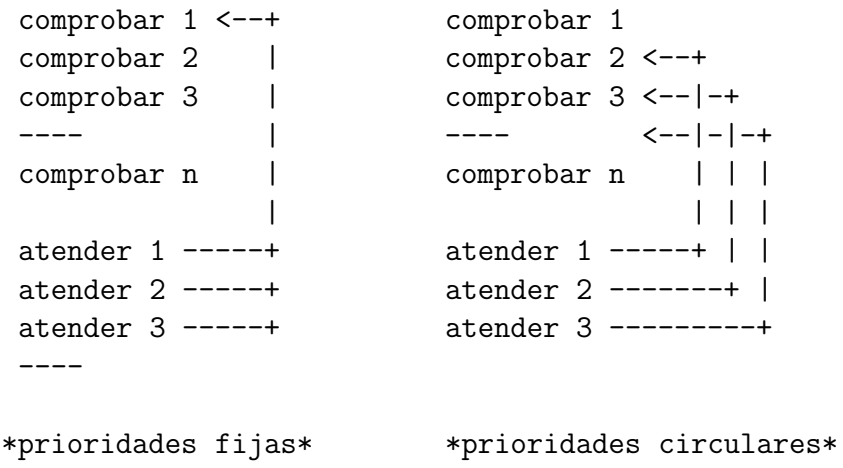
2.3. E/S programada

Un dispositivo se prepara [2] en el momento del arranque, o bien con posterioridad mediante el envío de instrucciones específicas. Una vez superada esta fase, la E/S controlada por programa se basa en el examen contínuo por parte del procesador del estado del dispositivo, hasta que está preparado para la siguiente transferencia.

Los inconvenientes de este método son la baja utilización del procesador y la dificultad para manejar múltiples dispositivos de E/S.

La E/S programada no tiene mucho interés práctico, ya que sólo puede soportar un dispositivo, y la utilización del procesador es muy baja, pues típicamente consume el 99 % del tiempo en un bucle de espera. Una técnica sencilla para mejorar algo la eficiencia es ampliar el bucle de manera que se comprueben varios dispositivos por turno rotatorio, y bifurcar a la rutina que corresponda al primer dispositivo que esté preparado.

[2] Está muy extendida la palabra *inicializar*, lo cual es una *traducción* incorrecta del inglés que debería ser rechazada. ¿O acaso decimos *finalizalizar*? Nosotros emplearemos preparar o iniciar.



Los retornos de estas rutinas pueden ajustarse de modo que se reproduzca un esquema de prioridades determinado. Por ejemplo, si todas vuelven al principio de la encuesta, el primer dispositivo tiene mayores probabilidades de ser atendido.

La noción de encuesta puede extenderse para proporcionar multitarea en un sistema de propósito general sin recurrir a interrupciones. El rendimiento puede mejorarse efectuando los test de E/S sólo *de vez en cuando*. Pero decidir cuanto es *de vez en cuando* es un tema difícil sin una respuesta sencilla, pues es fácil caer en alguno de los dos extremos: espera innecesaria o pérdida de datos.

2.4. E/S guiada por interrupciones

Las interrupciones son un mecanismo asistido por hardware para sincronizar el procesador con los sucesos asíncronos. Después de dar servicio a un dispositivo ejecutando una rutina de servicio de interrupción (RSI en adelante) el procesador reanuda la actividad en el punto en que se produjo la interrupción. Pero una RSI es un programa que modificará registros del procesador y posiciones de memoria. Por tanto, será preciso presevar el *contexto* del procesador en el momento de ser interrumpido, y recuperarlo justo después de que termine la RSI. Puesto que el programa interrumpido ignora cuando será interrumpido, es la RSI la encargada de salvar y restaurar el contexto.

Se han empleado distintos enfoques: desde salvar 'todo' por hardware hasta salvar sólo lo necesario por software.

A la llegada de una interrupción por una línea, el procesador no puede saber qué dispositivo la causó. Una opción es transferir el control siempre a la misma posición de memoria, donde se encuentra una rutina de encuesta. Pero entonces las probabilidades de ser atendidos no serían equitativas para todos los dispositivos, por lo que se prefiere recurrir a la vectorización de interrupciones. Esencialmente, la vectorización consiste en un enlace hardware entre el dispositivo que causa la interrupción y la RSI que lo atiende.

Para conseguir esto, el dispositivo que causa la interrupción debe proporcionar un identificador propio. La vectorización, dada su complejidad, requiere hardware adicional, y en concreto, suele usarse un controlador programable de interrupciones (CPI en adelante). La tendencia en los Sistemas Operativos modernos es a tener una tabla de vectores de interrupción bastante grande, y como aún así puede no ser suficiente para identificar unívocamente la fuente de una interrupción, puede combinarse este esquema con el de la encuesta.

Ahora bien, como pueden existir varias, o muchas, fuentes de interrupción, pueden existir varias peticiones sin atender en un momento dado. Pero la vectorización elimina la encuesta, y por eso debe proporcionarse algún método que permita asignar prioridades.

Citando textualmente la bibliografía:

> Un método es asignar un nivel diferente de prioridad a cada fuente única que puede ser vectorizada, o a grupos de dispositivos si el número de niveles distintos está más limitado. Esta ordenación es conocida normalmente por árbitros de prioridad hardware, que atienden a las peticiones en el orden de sus prioridades relativas (...) De este modo los periféricos pueden recibir servicio ajustado a sus exigencias y los dispositivos de altas prestaciones pueden operar a las velocidades mejores posibles.

Existen en general tres métodos de control de interrupciones:

1. Nivel general del sistema: las interrupciones pueden inhibirse/habilitarse de forma general. Esta opción es común a todos los sistemas informáticos (o casi todos).

2. Inhibición/habilitación de ciertas interrupciones. Se llama 'enmascaramiento' a esta técnica y se logra mediante un registro especial llamado 'máscara' cuyos bits indican qué interrupciones se encuentran habilitadas y cuales no. Se proporcionan medios para alterar los bits de esta máscara.

3. Nivel individual: manipulando registros específicos se puede deshabilitar la capacidad del dispositivo para interrumpir a la UCP.

2.5. Sobre la latencia de las interrupciones

Cuando se produce una interrupción, es preciso guardar el estado del procesador y posiblemente hacer algunas acciones administrativas adicionales que dependen de la implementación del sistema operativo. Por ejemplo, la asignación de valores a los semáforos que controlan el acceso a zonas de memoria compartidas entre el propio sistema operativo y las rutinas de servicio de interrupción. De la misma forma, cuando termina de ejecutarse una interrupción son precisas las acciones necesarias para llevar al sistema al estado previo a la interrupción. Todas estas operaciones consumen tiempo y limitan el número de interrupciones por segundo que un sistema operativo puede atender. En los sistemas operativos de propósito general, no está acotado el número de instrucciones que han de ejecutarse adicionalmente al propio servicio de interrupción para las tareas administrativas citadas y por tanto no está garantizado que el sistema responda en un tiempo acotado. Esa es la diferencia fundamental entre sistemas operativos de uso general, como Windows o Linux, y sistemas operativo de tiempo real, como QNX. En estos últimos se garantiza que la respuesta a una interrupción se producirá en un intervalo de tiempo fijo.

El hecho es que el tiempo invertido en acciones administrativas previas y posteriores a la ejecución de la rutina que atiende a la interrupción ha disminuido en proporción escasa comparada con el aumento de la velocidad de los procesadores, las memorias y los periféricos, y eso justifica la adopción no sólo de sistemas operativos específicos para operaciones de tiempo real (frecuentes en procesos industriales o sistemas de control), sino incluso la adopción de arquitecturas de procesador específicas para estas tareas. Estas reflexiones son más evidentes cuando se aplican a sistemas empotrados. Aquí, es importante no sólo que el tiempo de respuesta ante una interrupción sea pequeño, sino que además ese tiempo esté acotado. Véase la importancia de la arquitectura cuando se comparan algunos valores para procesadores es-

pecíficos de este tipo de sistemas. Un procesador Philips LPC2106, con una velocidad de reloj de 60MHz, sin caché pero con acelerador de memoria (año 2003-2004) emplea 432ns y 27 ciclos de reloj para entrar en una interrupción y 400ns y 25 ciclos de reloj para salir de ella. Pues bien, el procesador con arquitectura de pila RTX2000 de 1988, operando a 10MHz emplea 400ns y consume 0 instrucciones para entrar en una interrupción, y 200ns y una instrucción para salir. A pesar de que en los últimos 15 años la velocidad de reloj de este tipo de procesadores se ha multiplicado por un factor que varía entre 6 y 40 veces, no se ha mejorado el tiempo de latencia para atender interrupciones. Pero estos procesadores suelen tener arquitectura RISC. En procesadores CISC la situación es aún peor, pues pueden consumirse hasta 400 ciclos en la entrada a una interrupción. En otros trabajos [3] se ha comprobado la importancia de la arquitectura. Concretamente, al comparar un procesador RTX 2000 con un Sun 4 y un Sun 3 M68020 se encontró que el primero requiere para atender una interrupción tan sólo cuatro ciclos y 0.4μs, la máquina Sun 4 con procesador SPARC entre 200 y 400 ciclos y 14μs en el mejor de los casos y la máquina Sun 3 entre 1150 y 1600 ciclos, con un mejor caso de 81μs.

En cuanto al determinismo, los procesadores actuales emplean largos conductos (*pipes*) y grandes *caches* para aumentar la velocidad, pero esto no siempre es bueno porque tiene un impacto negativo sobre el determinismo. Por ejemplo, en sistemas que han de muestrear señales con una frecuencia determinada es importante que el muestreo se efectúe con la mayor regularidad posible.

Finalmente, otro factor a considerar es el comportamiento del sistema frente a las bifurcaciones en los programas. El análisis del código de un microcontrolador típico muestra que se produce control de flujo aproximadamente en una instrucción de cada 5, un 20 % del tiempo. Un Pentium IV con conductos de 20 etapas, *caches* y *buffers* de predicción de saltos puede perder 30 ciclos por cada fallo en la predicción de un salto. Un microcontrolador ARM pierde cuatro ciclos en el peor de los casos, y de nuevo vuelve a sorprender por su comportamiento totalmente determinista el procesador RTX2000 con arquitectura de pila, que pierde dos ciclos en cualquier caso. Aunque las cifras dadas para el Pentium IV dependen del *chip-set* y la memoria, se han

[3] *Real-time performance of the HARRIS RTX 2000 stack architecture versus the Sun 4 SPARC and the Sun 3 M68020 architectures with a proposed real-time performance benchmark*, William F. Keown, Philip Koopman y Aaron Collins. Performance evaluation review, vol. 19 #3, mayo de 1992

notificado retrasos de hasta 385 ciclos [4].

2.6. Consideraciones sobre el uso de *buffers*

Hemos dicho que una de las problemáticas de la E/S es la notable diferencia de velocidad entre la UCP y los dispostivos periféricos. Una de las formas de minimizar esta diferencia es mediante el uso de *buffers*, o zonas de almacenamiento temporal, donde la UCP coloca los datos en espera de que puedan ser enviados a un periférico que en ese momento puede estar ocupado. Evidentemente, la UCP puede llenar el *buffer* más rápidamente de lo que el periférico puede vaciarlo. Pero normalmente la UCP envía ráfagas de datos, de manera que entre una ráfaga y la siguiente el *buffer* puede haberse vaciado, al menos parcialmente.

En general, estamos ante un problema que puede modelizarse bastante bien mediante una cola. La UCP hace que aumente el tamaño de la cola, y el periférico hace que disminuya. En un momento dado, el estado de la cola es su longitud. Consideremos por simplicidad que la UCP envía bloques de caracteres de tamaño fijo, y que representamos el estado de la cola por el número de bloques que contiene.

Por simplicidad, supongamos en primera instancia que el *buffer* puede alojar un número infinito de bloques. Si llamamos l al número medio de bloques que llegan por unidad de tiempo, y m al número medio de bloques que el periférico puede procesar por unidad de tiempo (por ejemplo, imprimir), el número medio de transiciones entre un estado k y un estado $k+1$ es $lp(k)$, donde $p(k)$ es la probabilidad de que la cola contenga k bloques. De la misma forma, el número medio de transiciones entre un estado $k+1$ y un estado k es $mp(k)$.

En media, el número de transiciones en uno y otro sentido ha de ser igual, pues de lo contrario la cola crecería sin límite, o siempre estaría vacía. Entonces:

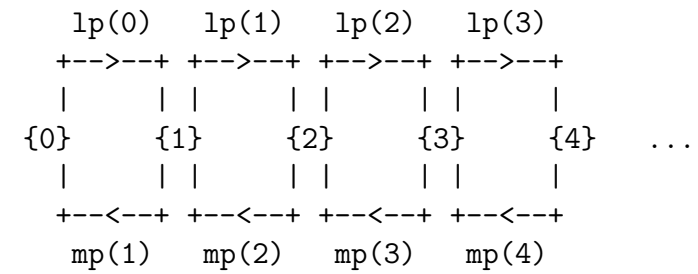
$$lp(0) = mp(1) \tag{2.1}$$

$$lp(1) = mp(2) \tag{2.2}$$

$$lp(2) = mp(3) \tag{2.3}$$

y así sucesivamente. Llamando $r = l/m$, de la primera:

[4]Datos tomados de *Programming Forth*, de Stephen Pelc, año 2005



$$p(1) = rp(0) \tag{2.4}$$

de la segunda:

$$p(2) = rp(1) = r^2 p(0) \tag{2.5}$$

y en general:

$$p(k) = r^k p(0) \tag{2.6}$$

Ahora bien, la suma para todo k, desde 0 hasta ∞, ha de ser la unidad, de donde se sigue que:

$$p(0) = 1 - r \tag{2.7}$$

y finalmente

$$p(k) = r^k (1 - r) \tag{2.8}$$

Entonces podemos calcular el número medio de bloques en la cola, N, que es:

$$N = \sum_{k=0}^{\infty} kp(k) = \frac{r}{1-r} \tag{2.9}$$

Si el *buffer* es finito, y sólo puede contener M bloques, entonces sigue siendo $p(k) = r^k p(0)$, pero ahora p(0) se sigue de la condición:

$$\sum_{k=0}^{M} kp(k) = 1 \tag{2.10}$$

de donde:

$$p(0) = \frac{1-r}{1-r^{M+1}} \tag{2.11}$$

Podemos usar la fórmulas anteriores para calcular la probabilidad de que la cola llegue a una longitud determinada, y por tanto podemos estimar el tamaño necesario del *buffer* para que la probabilidad de perder datos sea despreciable.

Ejemplo: Una impresora matricial en red dá servicio a una pequeña empresa, con cinco puestos desde los que se emiten facturas. Una factura tiene un tamaño de 2 bloques de 512 octetos. En media, durante una mañana laboral de 6 horas se emiten 200 facturas, que son 400 bloques, o 400/6 bloques/hora. Por su parte, la impresora puede imprimir un bloque cada 2 segundos, o 1800 bloques a la hora, lo que hace:

r=0.037
$p(0)$=0.963=96.3 %
$p(1)$=0.036= 3.6 %
$p(2)$=0.001= 0.1 %

es decir, con un *buffer* modesto, de 1 kb, la probabilidad de que se llene es de sólo el 0.1 %.

2.7. Otros modelos de colas

Supongamos el caso en que la UCP está enviando paquetes a varios periféricos. Cada periférico está asociado a una línea de salida. La UCP envía paquetes a un ritmo de l por segundo, y los periféricos se hacen cargo de los mismos a un ritmo de m por segundo. Identificamos el estado del sistema por el número total de paquetes en espera de tomar una línea de salida. En ese caso, la probabilidad del sistema de pasar de un estado k a un estado $k+1$ viene dada por lp_k, pero la probabilidad de pasar de un estado $k+1$ a un estado k es ahora $(k+1)mp_{k+1}$. Las ecuaciones del equilibrio son ahora

$$lp_0 = mp_1 \tag{2.12}$$
$$lp_1 = 2mp_2 \tag{2.13}$$
$$lp_2 = 3mp_3 \tag{2.14}$$
$$\tag{2.15}$$

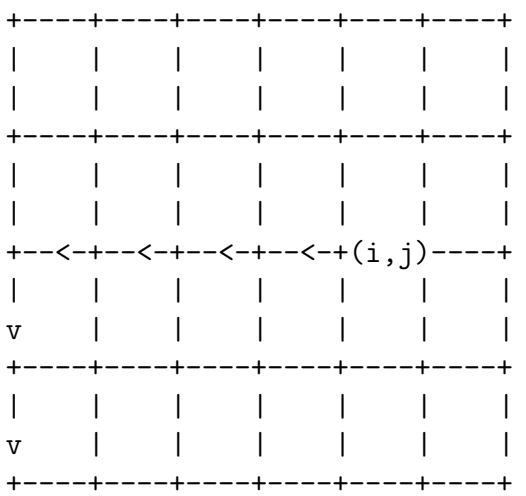
y así sucesivamente, de donde se sigue que

$$p_k = \frac{1}{k!}\left(\frac{l}{m}\right)^k p_0 \tag{2.16}$$

y de la condición de normalización:

$$p_0 = \left(\sum_k \frac{1}{k!}\left(\frac{l}{m}\right)^k\right)^{-1} \tag{2.17}$$

En general, los distintos procesos de espera, de envío y recepción de datos, tienen distintas prioridades. Por ejemplo, en una comunicación por red tiene mayor prioridad atender a la tarjeta de red que al teclado o la unidad de disco. Estas situaciones también pueden modelarse. La idea es que mientras que hasta ahora hemos modelado los distintos estados de una cola como un conjunto de puntos sobre una recta, cuando existen varias colas el proceso se visualiza mediante una malla en varias dimensiones. Por ejemplo, supongamos que existen dos colas con distintas prioridades:



Cuando el sistema se encuentra en un nodo de coordenadas (i, j) quiere decirse que existen i paquetes o clientes en una cola y j en la otra. Supongamos que la cola cuyos estados se corresponden con la coordenada horizontal tiene mayor prioridad que la cola cuyos estados se corresponden con la coordenada vertical. En ese caso, mientras i sea distinto de cero no podrá disminuirse j. Sólo después de vaciarse la cola de mayor prioridad podrá comenzar a vaciarse la cola de prioridad menor. Dicho de otra forma, desde el estado (i, j)

```
cs  - 1   28 - vcc
wr  -        - a0
rd  -        - INTA
d7  -        - ir7
d6  -        - ir6
d5  -        - ir5
d4  -        - ir4
d3  -        - ir3
d2  -        - ir2
d1  -        - ir1
d0  -        - ir0
c0  -        - INT
c1  -        - sp/en
gnd - 14  15 - c2
```

la cola sólo puede vaciarse siguiendo la trayectoria que se indica en la figura. Ahora, el sistema de ecuaciones de equilibrio es más complejo, pues ha de considerar los casos especiales $(0,0)$, $(i,0)$, $(0,j)$ e (i,j), y tener en cuenta dos probabilidades de entrada l_1 y l_2 y dos probabilidades de salida m_1 y m_2. La discusión de estos extremos nos llevaría demasiado lejos, pues el sistema resultante es difícil de resolver sin acudir a técnicas matemáticas especiales.

2.8. Gestión de interrupciones en el AT

2.8.1. Generalidades

El circuito integrado 8259 está diseñado para controlar las interrupciones. Sus principales registros son el llamado IRR (Interrupt Request Register) y el ISR (In Service Register). El IRR almacena todas las peticiones de interrupción pendientes, y el ISR todas las que están siendo atendidas en un momento dado. El 8259 dispone de una lógica para gestionar las prioridades relativas de las interrupciones que se produzcan, y por tanto para determinar qué interrupción ha de gestionarse primero.

Arriba se muestra un esquema del encapsulado del 8259.

Las lineas 'irx' estan directamente unidas a los periféricos que provocan las interrupciones. 'dx' constituyen un bus de datos, usado en la comunicación entre el 8259 y la UCP, en los dos sentidos. 'INT' es una línea directamente

unida a la correspondiente patilla en la UCP. 'INTA', por su parte, es necesaria para implementar un protocolo de comunicación entre el 8259 y el procesador, mediante el cual el primero comunica al segundo la llegada de una interrupción y el segundo acepta o rechaza atender a dicha interrupción. El 8259 puede funcionar en cascada, y mediante las lineas 'cx' pueden direccionarse desde un 8259 primario o maestro hasta 8 chips supeditados. 'rd' es la patilla de lectura. Cuando se activa, la UCP puede leer de los registros internos del chip. Analogamente, 'wr' es la patilla de escritura, y cuando está activada la UCP puede escribir en dichos registros. Estos dos últimos registros funcionan en conjunción con la patilla nombrada como 'cs'. Finalmente, 'sp/en' tiene que ver con los distintos modos de funcionamiento del 8259, ya que este gestor de interrupciones puede usarse en sistemas con distinta arquitectura al PC. Por supuesto, 'vcc' y 'gnd' son la fuente de voltaje (+5v) y el potencial de referencia.

En cuanto a los registros internos, tres son los fundamentales. Como se ha dicho antes, en el IRR se guardan, activando el bit correspondiente, las interrupciones que esperan ser atendidas. En el llamado ISR se guarda la interrupción que está siendo atendida en un momento dado. Finalmente, existe un registro especial que permite enmascarar interrupciones, de forma que éstas se atiendan o se ignoren, aunque físicamente se hayan producido.

En esencia, el 8259 funciona según la secuencia de acontecimientos siguiente:

1. Una o más líneas IRQ son activadas por los periféricos, lo que pone a 1 el bit correspondiente del IRR

2. El 8259 evalúa la prioridad de estas interrupciones y solicita la interrupción a la UCP

3. Cuando la UCP reconoce la interrupción, envía la señal '-INTA' al 8259

4. Al recibir la señal, el 8259 activa el bit correspondiente en el ISR y lo borra del IRR

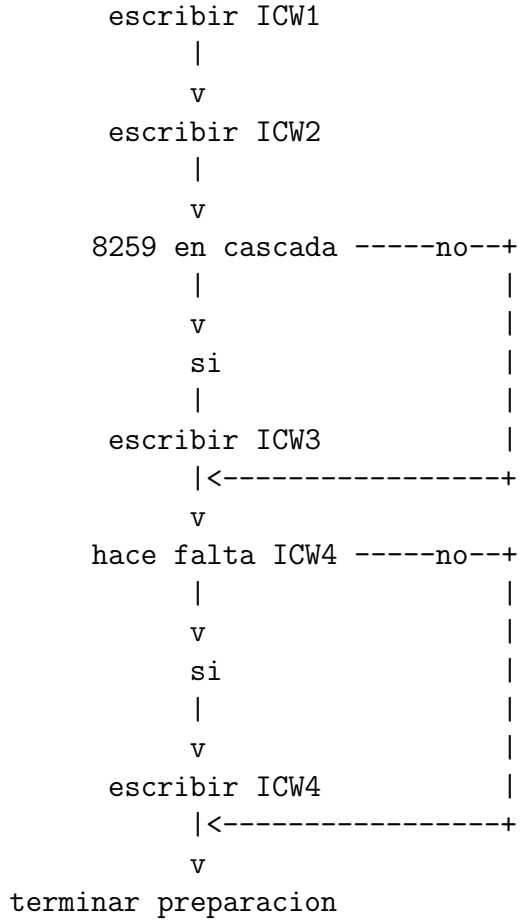
5. La UCP manda una segunda '-INTA' y el 8259 deposita en el bus de datos un valor de 8 bits que indica el vector de interrupción

6. En modo AEOI (Automatic End Of Interrupt), el bit del ISR es borrado en cuanto acaba la segunda 'INTA'. En modo EOI este bit ha de ser borrado explícitamente

Es interesante el punto 5. El 8259 sólo precisa 3 bits para indicar cual de las interrupciones se ha producido. Sin embargo, se envía un valor de 8 bits. Pues bien, los cinco bits más significativos indican el desplazamiento en la tabla de vectores de interrupciones que corresponde a la interrupción que se ha producido. Este valor es programable. Por ejemplo, durante el proceso de arranque del PC, el valor asignado es 8, lo que significa que la interrupción 'ir0', conectada al reloj, es atendida por la rutina apuntada por el vector de interrupción 8. La interrupción de teclado, conectada con el pin 'ir1', es atendida por la rutina apuntada por la entrada 9 de la tabla de vectores de interrupción, etc.

2.8.2. Preparación y control del 8259

El 8259 acepta dos tipos de órdenes generadas por la UCP: las ICW (Initialization Command Word) para preparación y las OCW (Operation Command Word) para programación.



El formato de los dos primeros ICW es el siguiente:

```
=ICW1=
0 0 0 1 L 0 S I -- info adicional en ICW4
^     ^ |   |
        |   +-- PIC en cascada
        +------ flanco o nivel

bit       valor      significado
===       =====      ===========
1          0/1       ICW4 innecesaria/necesaria
2          0/1       cascada/simple
3                    especifico del 8080/85
4          0/1       operacion por flancos/niveles
5          1
6                    especifico del 8080/85
7                    especifico del 8080/85
8                    especifico del 8080/85
```

El bit 5 a 1 y el 8 a cero le indican al 8259 que se trata de la primera palabra de preparación. Por otra parte, si el bit 1 se encuentra a 0, no será preciso enviar ICW3.

```
=ICW2=
x y z t u 0 0 0
---------
    |
    +--- offset en la tabla de vectores de interrupcion

bit       valor      significado
===       =====      ===========
0                    especifico del 8080/85
1                    especifico del 8080/85
2                    especifico del 8080/85
3          0/1       \
4          0/1       |
5          0/1        -> cinco bits altos del vector
6          0/1       |   de interrupcion a invocar
7          0/1       /
```

En cuanto a ICW3, es preciso distinguir entre primario y supeditado. Para el PIC primario, cada bit a 0 indica que la línea correspondiente está libre o conectada a un periférico que puede generar una interrupción. Un bit a 1 indica que esa línea está conectada con el PIC supeditado. En cuanto al PIC secundario, ICW3 contiene en los 3 bits de menor orden el número de identificación de dicho PIC.

Finalmente, y para nuestros propósitos, de ICW4 sólo nos interesa el bit 1. Cuando este bit se encuentra a 1, indica AEOI (Automatic End of Interrruption), lo que implica que el registro ISR es limpiado. Cuando este bit se encuentra a 0, indica que el registro ISR ha de limpiarse explícitamente.

Como ejemplo, considérese el siguiente fragmento de código, que se ocupa de preparar los PIC primario y secundario en un PC. Véase que al PIC primario se accede a través de los puertos 0x20 y 0x21, mientras que al PIC secundario se accede a través de los puertos 0xa0 y 0xa1. Además, es necesario saber a qué puertos se envían tanto los ICW como los OCW. Este detalle se recoge en la tabla inserta antes del código:

```
0x20 0xa0   ICW1
            OCW2
            OCW3
0x21 0xa1   ICW2
            ICW3
            ICW4
            OCW1
```

```
/* preparacion de los PIC en el PC */
outportb(0x20,0x11);  flancos/cascada/ICW4
outportb(0x21,8);     offset en Tabla Vectores Interrupcion
outportb(0xa1,0x70);  offset TVI secundario
outportb(0x21,4);     esclavo conectado a IRQ2
outportb(0xa1,2);     identificacion del secundario como 2
outportb(0x21,1);     ICW4, EOI
outportb(0xa1,1);
```

En cuanto a los OCW, este es el formato de OCW1 y OCW2:

```
=OCW1= Cada  uno de  los bits de  esta orden indica  si la
interrupcion asociada esta  enmascarada (1) o no enmascarada
(0)
```

```
=OCW2=
bit       valor      significado
===       =====      ===========
0          0/1       \
1          0/1        -> IR sobre la que actuar
2          0/1       /
3          0
4          0
5          0/1       \
6          0/1        -> 001/011 EOI
7          0/1       /
```

OCW3 está dedicado a funciones específicas (ver bibliografía) que alteran el esquema normal de prioridades. Los dos primeros bits son interesantes. Cuando toman el valor 10 indican que se procede a leer el registro IRR. Cuando toman el valor 11 indican que se procede a leer el ISR.

Como ejemplo, la siguiente línea de código deshabilita la interrupción de reloj:

```
outportb(0x21,1);
```

y las dos líneas siguientes leen qué interrupciones están pendientes de ser procesadas:

```
outportb(0x20,0xa0); OCW3 al primario
IRR=inportb(0x20);
```

2.8.3. Implantación

Como se ha dicho, en los AT y posteriores, existe un segundo 8259 conectado en cascada al primero. Al primero se accede mediante los puertos 0x20 y 0x21, y al segundo mediante los puertos 0xa0 y 0xa1. Al arrancar, la BIOS coloca la base de interrupciones del primer controlador a 8, con lo que IR0-7 quedan ligadas a los vectores 8-15. El segundo 8259 genera las interrupciones 0x70 a 0x77. La asignación de los diversos IR es la siguiente:

```
IRQ0            temporizador
IRQ1            teclado
IRQ2            E/S en los XT
```

```
  IRQ8          reloj tiempo real
  IRQ9          simulacion de la IRQ2 del XT
  IRQ10         reservado
  IRQ11         reservado
  IRQ12         reservado
  IRQ13         coprocesador
  IRQ14         disco duro
  IRQ15         reservado
IRQ3            COM2
IRQ4            COM1
IRQ5            paralelo 2
IRQ6            disketeras
IRQ7            paralelo 1
```

Existe una interrupción no enmascarable que se produce cuando se detecta un error de paridad en la memoria. Sin embargo, incluso esta interrupción puede enmascararse en los AT, poniendo a 1 el bit 7 en el puerto 0x70.

2.8.4. Ejemplos

Si se escribe una rutina de servicio para la interrupción de temporizador, por poner un ejemplo, tenemos dos opciones: gestionar completamente la interrupción o efectuar un tratamiento parcial específico, llamando a continuación a la rutina original. En el primer caso, al terminar es preciso hacer un EOI enviando OCW2 con valor 0x20 al puerto 0x20.

El siguiente programa está escrito en Turbo C y muestra como puede capturarse la interrupción de teclado, desviándola a una rutina propia. Esta rutina se limita a modificar una variable llamada `testigo`. La función principal se ocupa de las modificaciones en los vectores de interrupción y después entra en un bucle, esperando simplemente a que `testigo` tome el valor del código de rastreo de la tecla F1. En la práctica, esta espera podría dedicarse a efectuar cualquier cálculo de forma concurrente con la entrada desde teclado. Cuando se produce la condición de salida, se restaura el vector de interrupción original y se sale del programa. Obsérvese el modificador `interrupt` en la función `mirutina()`, que informa al compilador de que produzca código específico para guardar los registros de la CPU y para restaurarlos a la salida, y de que incluya al final de la misma una instrucción `iret` en lugar de la `ret` habitual. Obsérvese también que puesto que `mirutina()` será invocada por hardware, no pueden pasársele argumentos, y de que no es llamada por ninguna función distinta dentro del programa original.

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

int testigo=10;
char *p=(char *)MK_FP(0xb800,0);

void interrupt mirutina()
{
testigo=inportb(0x60);
*p=testigo;
outportb(0x20,0x20); /* EOI */
}
main()
{
int *p=(int *)MK_FP(0,36);
int des,seg;

/* guarda vector de interrupcion   */
des=*p;
seg=*(p+1);

/* cambia vector de interrupcion   */
asm cli;
*p=FP_OFF(mirutina);
*(p+1)=FP_SEG(mirutina);
asm sti;

/* espera a que se pulse la tecla F1   */
while (testigo!=59); /* scancode de F1 */

/* restaura vector de interrupciones   */
asm cli;
*p=des;
*(p+1)=seg;
asm sti;
printf("terminado %d\n",testigo);

}
```

Capítulo 3

El teclado

3.1. Secuencia de sucesos

Que duda cabe de que, junto con el adaptador de vídeo, el teclado es el periférico más universal. En el presente capítulo estudiaremos su funcionamiento y control. Pero, antes de eso, conviene repasar brevemente el camino que se recorre desde que se pulsa una tecla hasta que se ejecuta la acción correspondiente, que puede ser desde representar en pantalla un símbolo ASCII hasta generar un sonido o ejecutar una rutina específica.

La secuencia de sucesos que se produce es la siguiente:

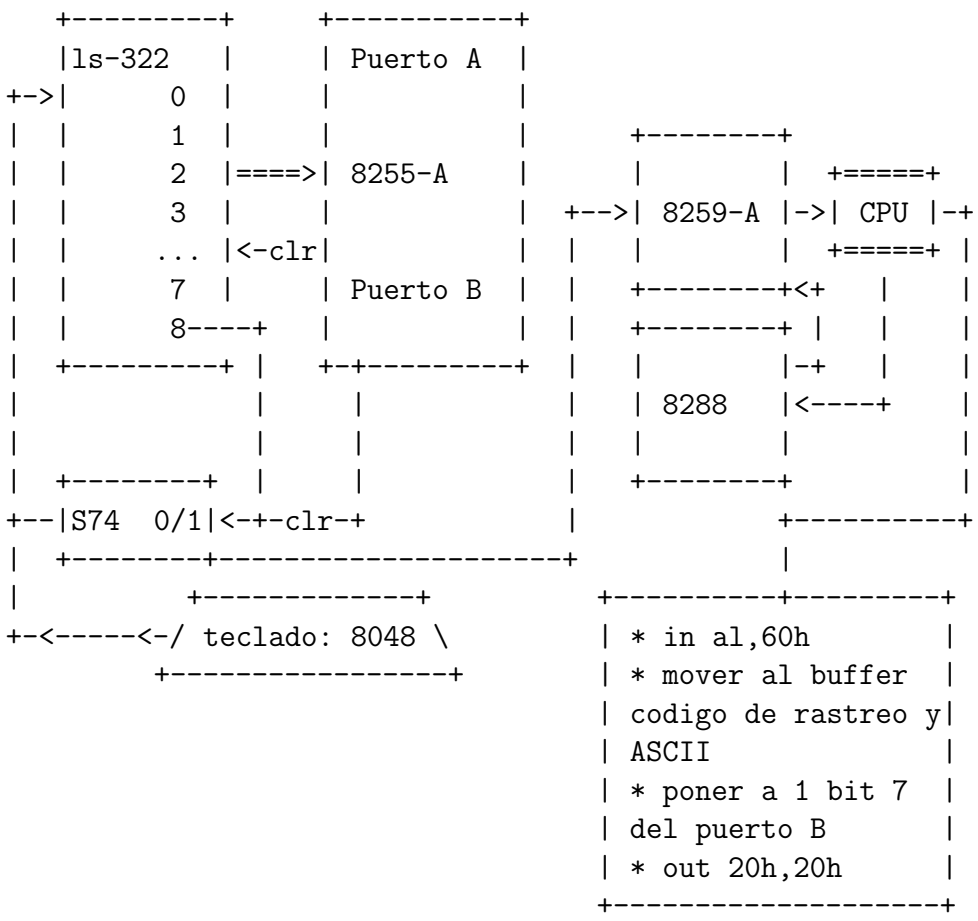
1. El 8084 es un micro de 8 bits, con 64 octetos de RAM, una ROM y una BIOS, que recibe el código de rastreo del teclado. Este código depende de qué tecla se pulse, pero *no* del carácter o símbolo que la tecla tenga grabado sobre su superficie.

2. El 8048 pide permiso al S-74. Si lo obtiene, pasa el código en serie al LS-322. Si no lo obtiene, pasa el código a un *buffer* interno. Si el *buffer* está lleno, manda una señal 0xff al S-74 y se produce un pitido.

3. El bit de inicio usado en el paso anterior es empujado fuera del registro de desplazamiento del LS-322.

4. El S-74 bloquea al 8084 y pone a 1 el bit correspondiente a la IRQ1 en el registro del PIC.

5. El PIC pasa la demanda de interrupción a la CPU, que la atenderá siempre que se encuentre en estado STI.


ls-322
0
1
2
3
...
7
8
S74
0/1
clr
clr
Puerto A
Puerto B
8255-A
8259-A
CPU
8288
teclado: 8048
* in al,60h
* mover al buffer
codigo de rastreo y
ASCII
* poner a 1 bit 7
del puerto B
* out 20h,20h

6. Si la atiende, comunica al PIC la aceptación de la interrupción a través del controlador del bus 8288.

7. Esta aceptación actúa como temporizador, e inicia otra serie de acciones: el PIC envía el número 9 a la CPU y activa el bit correspondiente en el ISR.

8. La CPU usa el numero 4*9 como entrada en una tabla y:

   a) hace CLI
   b) guarda CS
   c) carga nuevos CS:IP
   d) hace STI
   e) pasa el control a la nueva dirección, donde se ejecuta el código que atiende a la interrupción. Este código esencialmente lee el puerto donde se encuentra el código de rastreo, lo traduce a código ASCII y lo pone en el *buffer* del teclado.
   f) envía un EOI al PIC
   g) hace un IRET [1]

9. Se pone a 1 el bit 7 del puerto B, que está conectado a la línea 'clr' del LS-322 y del S-74

3.2. Tipos de teclados

Recordemos que no importa el carácter que se encuentre grabado sobre la tecla, sino el código de rastreo que se produce al pulsarla. Así, aunque hay teclados para usuarios suecos, ingleses, españoles o franceses, en realidad sólo existen, en el ámbito del PC, tres teclados diferentes: el original PC/XT de 83 teclas, el teclado AT de 84 teclas y el AT de 101 o 102 teclas. La llegada de los ordenadores portátiles ha provocado una gran variedad de teclados en los últimos tiempos, aunque al programador esta variedad no debe preocuparle, ya que o bien emulan internamente alguno de los teclados anteriores, o bien transforman sus códigos de rastreo. Citemos por último los teclados con *track- ball* incorporado o teclas que permiten mover y accionar el puntero del ratón. Nuevamente, estas innovaciones son transparentes para el programador, ya que las teclas responsables son controladas por el controlador de ratón habitual. Se remite al lector a la bibliografía para los detalles. Por

[1] Interuption RETurn

último, cabe citar la importancia que últimamente se le esta concediendo a los aspectos ergonómicos del teclado, intentando que las manos descansen en una posición natural. En nuestra opinión, el uso intensivo de cualquier instrumento puede acarrear lesiones, y en el caso del teclado no es tan importante la disposición de las teclas, que suelen dividirse en dos grupos separados en los teclados *ergonómicos*, como la disposición del teclado sobre la mesa, que no debe colocarse en el borde de la misma sino a unos centímetros de dicho borde, facilitando el apoyo de las manos. El lector hará bien en distinguir entre verdaderas ventajas ergonómicas y argumentos puramente comerciales. En cuanto al tacto del teclado, tiene una gran importancia en la comodidad del usuario, y ya que es el periférico con el que se tiene un contacto más directo, conviene elegir uno de calidad. En general, los teclados de membrana, más baratos, tienen un tacto suave, pero poco preciso, mientras que los teclados mecánicos tienen un tacto algo más duro pero también una mayor precisión, que se agradece cuando se usa el teclado durante mucho tiempo. Los mejores teclados combinan suavidad y precisión, y garantizan muchos millones de pulsaciones por cada tecla.

3.3. El teclado perfecto

Continuemos nuestra discusión sobre los aspectos ergonómicos relacionados con el teclado. Existe una abundante literatura sobre el teclado *Dvorak*. Este teclado se distingue del teclado tradicional por la disposición de sus teclas. La historia, repetida una y mil veces, es que en el teclado tradicional la disposición de las teclas es aquella que garantiza la *menor* velocidad de tecleo. Esta característica es una herencia de los teclados de las antiguas máquinas de escribir. En aquellas máquinas, cada tipo estaba grabado sobre una palanquita, que al ser accionada pulsando la tecla correspondiente golpeaba una cinta impregnada de tinta tras la cual se encontraba el papel. Por tanto, una de estas palanquitas, por su propia inercia, necesitaba un tiempo para retirarse del papel. Una alta velocidad de tecleo podía hacer que unas palancas golpeasen con otras, y por ese motivo las letras que en los textos aparecen (en inglés) formando parejas con frecuencia se separaron lo más posible, para que así unas palancas no interfiriesen con otras. Un efecto de este diseño es que la velocidad de tecleo no puede ser tan alta como en teoría es posible. Otro, que no existe una buena alternancia entre las pulsaciones de una mano y otra. En el teclado *Dvorak* la disposición de las teclas es tal que la velocidad de tecleo de los digrafos más frecuentes en inglés es máxima. Se sigue por tanto que no existe un único teclado *Dvorak*, sino uno adaptado para cada idioma.

Existe desde hace décadas una discusión, a nuestro juicio estéril, sobre el porqué de la no adopción universal de un teclado que en teoría y según sus defensores en la práctica es superior. Sin negar esta superioridad funcional, seguramente intervienen otros muchos factores. En primer lugar, el teclado tradicional no es manifiestamente malo, y la velocidad de tecleo que se puede conseguir con él es más que aceptable. En segundo lugar, la ventaja se reduce si en lugar de medirla sobre una máquina de escribir mecánica se mide sobre el teclado de un ordenador, como es el caso desde hace un par de décadas. En tercer lugar, la inercia de la masa de usuarios es enorme. Quizás muchos adoptasen un teclado *Dvorak* si estuviesen seguros de que iba a ser el teclado universal, pero a nadie le agrada tener que aprender dos teclados. En cuarto lugar, la alta velocidad de tecleo puede ser prioritaria en ciertas profesiones, pero no para la mayoría.

Todas estas objecciones sin embargo no han hecho disminuir la discusión. De entre todo el ruido resultante, destacan los experimentos encaminados a conseguir un *teclado perfecto*. Estos experimentos demuestran que el teclado *Dvorak* es uno más, y no el mejor, de los muchos que pueden diseñarse. El que hemos llamado teclado tradicional fue diseñado para las máquinas de escribir de la marca Remington en 1876 por Christopher Sholes, mientras que el teclado Dvorak fue creado en 1930. En ninguno de los casos se dispuso, claro está de la ayuda de ordenadores. Puede aducirse que no es necesaria la ayuda de un ordenador para situar unas pocas decenas de teclas, y en efecto, pero sí si lo que se busca es la distribución óptima. En los experimentos para conseguir un teclado perfecto se parte de un teclado de treinta teclas: las veintiseis del alfabeto inglés más unos pocos signos de puntuación. A continuación, se diseña una función de coste que mida el esfuerzo necesario para teclear una palabra, penalizando algunas situaciones y bonificando otras. Por ejemplo, tiene buena puntuación pulsar dos teclas distintas sin necesidad de mover los dedos de la fila central, y mala puntuación tener que pulsar dos teclas consecutivas con el mismo dedo. Los desplazamientos de uno o varios dedos, también penalizan en mayor o menor medida. El siguiente paso consiste en reunir unas cuantas decenas de megabytes de texto significativo. En el caso que estamos describiendo, se tomaron media docena de obras de la literatura inglesa, varios años de correo electrónico y unas cien mil lineas de código fuente en C. A continuación se programa un entorno evolutivo donde se comienza asignando aleatoriamente la disposición de las teclas para 4K teclados distintos. Un programa lee los textos de prueba, y con la función de coste se calcula, para cada teclado, el coste de teclear los textos. Se eliminan los 2K teclados que obtuvieron una puntuación inferior y se rellena la tabla

hasta completar de nuevo los 4K teclados produciendo alteraciones aleatorias sobre los 2K teclados mejores de la ronda anterior. Se repite el proceso hasta que aparece un ganador estable. Este ganador se almacena aparte, se toma otro conjunto de 4K teclados aleatorios y se repite el proceso anterior, obteniendo tras una serie de rondas un segundo ganador, y luego un tercero, etc. En una ronda final se hacen competir los ganadores de las rondas anteriores, junto con el teclado "qwerty" y el teclado *Dvorak*. Estos son los teclados "qwerty", *Dvorak* y el teclado ganador, junto con sus puntuaciones:

```
' , . p y  f g c r l      Dvorak
a o e u i  d h t n s      12,189,785
; q j k x  b m w v z

q w e r t  y u i o p      Sholes
a s d f g  h j k l ;      25,390,660
z x c v b  n m , . '

. u y p q  k l d c g      Mejor teclado
e a i n w  r h t s o      9,640,479
' , ; f z  j m v b x
```

El experimento no termina aquí. Una vez conseguido un teclado ganador, se usó durante un tiempo para probar su ergonomía, y se comprobó que en realidad la función de coste era mejorable. Se simplificó algo y se repitió el experimento, obteniendo un teclado ganador algo más cercano al óptimo:

```
' , . p y f g c r l      Dvorak
a o e u i d h t n s      32,129,548
; q j k x b m w v z

q w e r t y u i o p      Sholes
a s d f g  h j k l ;      59,514,344
z x c v b  n m , . '

k , u y p w l m f c      Mejor teclado
o a e i d r n t h s      28,281,895
q . ' ; z x v g b j
```

Aún puede mejorarse algo. Puede escribirse un programa que recopile toda la información de tecleo de una persona en un uso normal del teclado durante unas cuantas semanas. Esta información incluye medidas de tiempo. Si se

divide el teclado en seis grupos de teclas, tal y como se ha representado arriba, se puede calcular una matriz de transición de un grupo a otro, y construir a partir de aquí una función de coste experimental.

Por otra parte, el tecleo de grandes cantidades de texto se produce en un editor de texto. Sería posible, usando un razonamiento similar al que hemos presentado, construir un esquema evolutivo que permitiese encontrar el *editor perfecto*, introduciendo de paso un factor más de discusión tanto para los acérrimos defensores de *vi* como para los no menos aguerridos defensores de *emacs*. Hasta donde sabemos, ese experimento aún no se ha realizado, aunque sí se han recolectado datos experimentales y se dispone de las matrices de transición [2]

3.4. Acceso al teclado a través de la BIOS

Existen tres funciones de la BIOS que se encargan del control de teclado en los programas de usuario, pero que no tienen en cuenta las mejoras introducidas con el teclado extendido de 102 teclas; por ejemplo, no reconocen las teclas de funcion `F11,12`. El código ASCII generado por el controlador de la ROM-BIOS o KEYB puede obtenerse mediante dos funciones de la interrupción 0x16.

La funcion 0x00 lee un carácter del *buffer* de teclado. Si no hay ninguno almacenado, espera hasta que se introduzca uno. El caracter leído se elimina del buffer del teclado.

La función 0x01 determina si hay un carácter en el *buffer* del teclado, devolviéndolo a la función invocadora si es así. A diferencia de la función anterior, el carácter no se elimina del *buffer*, de forma que puede ser llamado repetidamente. Consúltese la bibliografía para obtener los detalles de las llamadas a las funciones de la BIOS. En general, ciertos valores son colocados en registros específicos, se llama a la interrupción encargada de ejecutar la función y el resultado queda de nuevo en registros específicos. Cada función de cada interrupción es descrita en la documentación, por lo que no tocaremos aquí este punto.

[2] Véase *How do people really use text editors?*, J. Whiteside, Norman Archer, Dennis Wixon y Michael Good; Digital Equipment Corporation. SIGOA Newsletter, 3; Junio 1982

Como ejemplo, veamos como se podría usar la función anterior para escribir una rutina de impresión que permitiese al usuario detener la impresión en un momento dado pulsando la tecla ESC. Escribamos la rutina en pseudo-código:

```
IF [todos los caracteres impresos] FIN;
ELSE {
  [imprimir siguiente caracter]
  [llamar a la funcion 01h de int 16h]
  IF ![flag de cero]{
       [obtener caracter mediate funcion 00H]
       IF [caracter==ESC] FIN;
       ELSE [continuar imprimiendo]
  }
}
```

La funcion 0x02 permite encontrar el estado de los diferentes modos del teclado, así como la posición de determinadas teclas de control. Ocurre que la BIOS dispone de un rango de direcciones en memoria convencional donde depositar ciertas informaciones. En concreto, el estado del teclado se representa mediante un par de octetos en esta zona, y la función que nos ocupa permite consultarlos.

Con la llegada del AT se añadieron algunas nuevas funciones, como control de factor de repetición, para fijar el ritmo de repetición del teclado cuando una tecla se mantiene pulsada, simulación de pulsación de una tecla y, desde luego, control del teclado ampliado. La funcion 0x12 devuelve la misma información que la 0x02 en AL, pero además devuelve en AH la información adicional siguiente:

```
bit  valor significado
0     1    Ctrl pulsada
1     1    Alt pulsada
2     1    Pet Sis pulsada
3     1    Modo de pausa activado
4     1    Tecla Inter pulsada
5     1    Bloq Num pulsada
6     1    Bloq Mayus activado
7     1    Insert activado
```

Especialmente interesante es la función que permite simular la pulsación de una tecla, pues puede aprovecharse para elaborar macros de teclado.

Cuando una de las funciones de la BIOS devuelve una tecla en AL, coloca en AH el código de rastreo de la tecla, pero en el teclado ampliado, cuando se pulsa una tecla especial, como una tecla de función o Ctrl, por ejemplo, en AL se devuelve el código ASCII 0, mientras que la información que nos interesa, el código de tecla ampliado, se encuentra en AH. En los lenguajes de alto nivel Pascal o C, las funciones `readkey` o `getch()` devuelve el código ASCII. Cuando éste es cero, una segunda llamada devuelve el código extendido. Esta es la clave de las rutinas de entrada desde teclado. El siguiente programa ilustra la situación:

```
#include <stdio.h>
#include <conio.h>
main()
{
    int c;
    do{
        if (!(c=getch())) c=getch();
        printf("%d\n",c);
    } while (c!=13); /* repetir hasta que se pulse INTRO */
}
```

En lenguajes como BASIC, la situación es ligeramente diferente. El siguiente programa toma entradas de teclado y presenta el resultado hasta que se pulse la letra 'x':

```
CLS
DO
tecla$ = INKEY$
IF LEN(tecla$) = 1 THEN
     PRINT ASC(tecla$)
ELSEIF LEN(tecla$) = 2 THEN
     tecla$ = RIGHT$(tecla$, 1)
     PRINT ASC(tecla$)
END IF
LOOP UNTIL tecla$ = "x"
```

La funcion `INKEY$` devuelve una cadena, de longitud simple o doble según que se haya pulsado una tecla normal o una tecla extendida.

3.5. El controlador de teclado

Desde la introducción del AT, es posible transmitir información hacia el teclado. Este posee un registro de estado y dos *buffer*s, uno de entrada y otro de salida. El de salida se emplea para los códigos de teclado y los datos que el sistema haya solicitado previamente al teclado mediante una orden. A este *buffer* se accede mediante la dirección de puerto 0x60. Por otra parte, al buffer de entrada puede accederse tanto a través de la dirección de puerto 0x60 como a través de la 0x64.

Si se desea transmitir una orden al teclado, usamos el puerto 0x60 mientras que el octeto de datos asociado se envía a través de 0x64. En particular, cuando una tecla se encuentra pulsada, el bit 7 del registro de estado se pone a cero. El siguiente fragmento, tomado de la ayuda en línea de Turbo Pascal, muestra como se puede mejorar el funcionamiento del teclado consultando el puerto 0x60 y emitiendo un breve *biip* cuando se pulsa una tecla. Muchos mecanógrafos prefieren este sistema, ya que así tienen además de la confirmación táctil una confirmación auditiva de la pulsación.

```
procedure Keyclick; interrupt;
begin
    if Port[$60] < $80 then
    { Solo cuando hay una tecla pulsada >}
    begin
        Sound(5000);
        Delay(1);
        Nosound;
    end;
end;
```

Dos de los controles más útiles que pueden realizarse sobre el teclado son el factor de repetición y el tiempo de retardo. El primero indica a cuantas pulsaciones por segundo equivale el mantener pulsada una tecla, es decir, la velocidad a la que aparece en pantalla la secuencia *aaaaaa...* cuando se mantiene pulsada la tecla *a*. Esta función sólo comienza a ejecutarse un cierto tiempo después de que la tecla ha sido pulsada, evitando de esta forma repeticiones no deseadas de teclas. Ambos parámetros, factor de repetición y tiempo de retardo, podríamos ajustarlos enviando las órdenes y datos adecuados a los puertos correspondientes. Sin embargo, el envío de datos u órdenes al teclado no es directo, y requiere de un protocolo algo complicado, que omitiremos.

Es mucho más sencillo usar las funciones BIOS. La función 0x03 permite establecer la frecuencia de repetición y el tiempo de retardo. Para utilizar este servicio, se llama a INT 0x16 con las siguientes entradas:

```
ENTRADA: AH = 0x03
     AL = 0x05
     BL = frecuencia deseada(*)
     BH = retardo(**)


(*) Varia de 30 a 2 caracteres por segundo:
00H   30.0    0BH   10.9    16H    4.3
01    26.7    0C    10.0    17     4.0
02    24.0    0D     9.2    18     3.7
03    21.8    0E     8.6    19     3.3
04    20.0    0F     8.0    1A     3.0
05    18.5    10     7.5    1B     2.7
06    17.1    11     6.7    1C     2.5
07    16.0    12     6.0    1D     2.3
08    15.0    13     5.5    1E     2.1
09    13.3    14     5.0    1F     2.0
0A    12.0    15     4.6    20     Reservado

(**)
00H   250 ms
01    500
02    750
03    1000
```

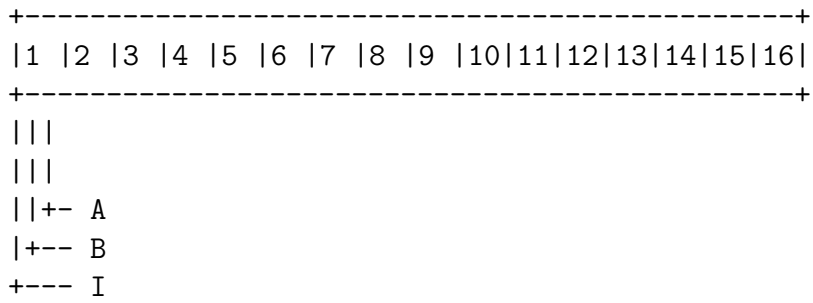
Ya que estamos mencionando las funciones BIOS, ilustraremos el uso de las mismas mediante un ejemplo, donde se establecerá un tiempo de retardo de 250 milisegundos. La mayoría de los compiladores modernos tienen predefinidas unas variables que actúan como pseudoregistros, de manera que a través de ellos se pueden cargar los valores que después requieren las interrupciones software.

```
#include <stdio.h>
#include <dos.h>
main()
{
    union REGS pseudo; /* pseudoregistros */
```

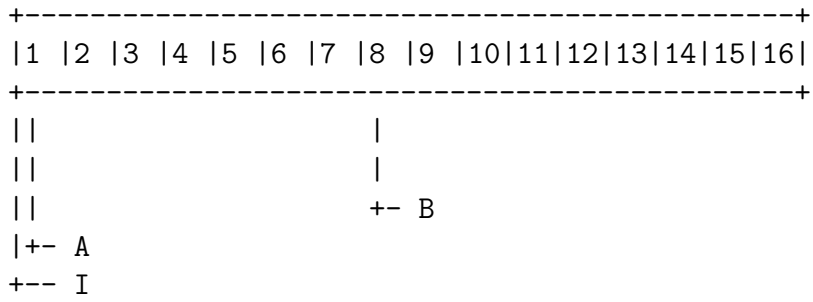
```
    pseudo.h.al=0x05;
    pseudo.h.ah=0x03;
    pseudo.h.bl=0;
    pseudo.h.bh=0;
    int86(0x16,&pseudo,&pseudo);
    return(0);
}
```

3.6. Estructura del buffer de teclado

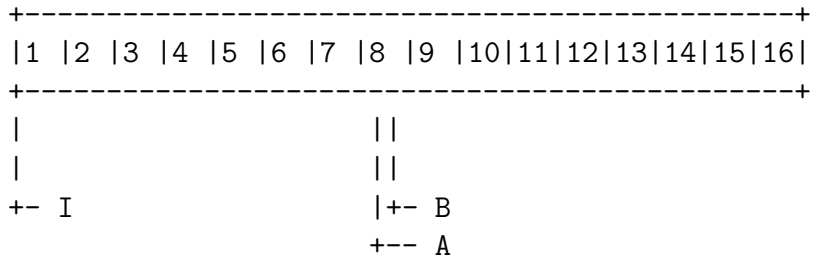
Como sabemos, existe una zona de datos de la BIOS en RAM que contiene información sobre el estado del teclado. La rutina de la interrupción de teclado usa esta zona como *buffer* que después puede ser consultado por las funciones de la BIOS. Describiremos la estructura de este *buffer*. En principio, podríamos diseñarlo como una pila FIFO. Reservaríamos una serie de octetos para la pila y allí se depositarían los caracteres. Cuando se leyese un carácter, se extraería de la parte inferior de la pila y el resto de los valores se desplazarían una posición hacia abajo, quedando disponible en la base de la pila el siguiente carácter a leer. Para escribir un carácter, se comprobaría que la pila no sobrepasase el tamaño asignado y se escribiría en la parte alta. Este esquema, con ser válido y lógico tiene el inconveniente de que obliga a mover el contenido íntegro de la pila cada vez que se lee un carácter. Por eso se recurre a una solución distinta, que consiste en usar un *buffer* circular con dos punteros, uno hacia el siguiente carácter a leer y otro hacia la siguiente posición para escribir. Cuando se actualiza la pila en una operación de lectura o escritura, sólo es preciso modificar uno de los dos punteros, con el ahorro de código y tiempo. Se necesita un tercer puntero para marcar el comienzo del buffer en la memoria. Este puntero tiene el valor 0040:001E. A partir de este punto se reservan dieciseis palabras, correspondientes a las 16 teclas que admite por defecto el *buffer* del teclado. Cada palabra tiene un octeto alto y un octeto bajo. Si la tecla es normal, el octeto bajo contiene el código ASCII de la tecla, y el octeto alto el código de rastreo. Si la tecla es especial, el octeto bajo contiene el valor cero, y el octeto alto el código de teclado extendido. Supongamos que partimos de una situación en que el *buffer* se encuentra vacio. Llamaremos I al puntero que marca el inicio del *buffer*, A al puntero que marca el siguiente caracter a leer y B al que marca la siguiente posición a escribir. En el estado original, I=A=B.



Si se escriben siete caracteres, la situación es la siguiente:



Si se leen siete caracteres, la disposición de los punteros es como sigue:



Por tanto, cuando A y B coinciden, significa que el *buffer* se encuentra vacio. Cuando alguno de los dos punteros apunta a la posición 16, en su

siguiente actualización pasará a apuntar a la posición 1. El único conflicto que puede aparecer es cuando B apunta a la palabra anterior a la que apunta A. En ese caso una operación de escritura llena el *buffer*.

3.7. El teclado y el antiguo DOS

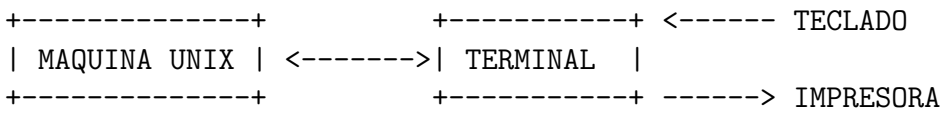
Existe una literatura profusa sobre las técnicas de programación de teclado en DOS. En concreto, floreció durante mucho tiempo la creación de los llamados *programas residentes*. DOS no es un sistema multitarea, pero se desarrollaron técnicas para tener en memoria más de un programa simultáneamente. Estos programas *dormidos* eran invocados a través de interrupciones. El hecho de que cualquier programas en DOS pudiese apoderarse de las interrupciones pronto derivó en incompatibilidades, falta de estabilidad y varios intentos infructuosos de establecer unas normas de convivencia entre programas. Aunque también hubo guerras declaradas entre programas rivales. En versiones anteriores de estos apuntes se incluían programas de ejemplo para ilustrar estas técnicas, que hoy en día creemos que han perdido todo su interés.

Capítulo 4

Terminales

4.1. Introducción

El concepto de terminal proviene de la época de los grandes computadores centrales, que debían dar servicio a un número variable de usuarios. Cada uno disponía de un terminal, formado por teclado, monitor y tal vez impresora, mediante el cual se comunicaba con el ordenador central, en donde residía toda la potencia de cálculo, y a su vez recibía información de retorno. Con el tiempo, los terminales fueron dotados de la lógica necesaria para interpretar ciertas secuencias de octetos, y efectuar operaciones relativamente complejas de control del monitor, como desplazamientos, borrado selectivo, ubicación del cursor, etc.



El aumento de la potencia de cálculo disponible localmente para cada usuario, especialmente desde la llegada del PC, ha relegado progresivamente el uso de terminales, pero hay varias razones por las que vuelven a ser una posibilidad sumamente interesante. En primer lugar, la llegada del entorno UNIX al ordenador personal ha traído de nuevo el concepto de terminal, bien en su modalidad de terminal virtual, sirviendo de interfaz entre el usuario y el sistema, bien en su modalidad clásica, conectado a través de un puerto serie o paralelo. En segundo lugar, el concepto de concentrar en una sola máquina la potencia de cálculo es interesante porque reduce drásticamente los costes y el mantenimiento.

Considérese por ejemplo el caso de una pequeña empresa que cuenta con cinco puestos basados en ordenadores personales que forman una red local. Los costes de un equipamiento similar son ciertamente elevados. Ahora bien, dada la potencia actual de los procesadores, y la capacidad de discos duros y memorias, un PC modesto puede dar servicio a cinco usuarios sin ningún problema. Estos pueden conectarse a través de un obsoleto 2/386 con un programa de emulación de terminal (uno de los mejores, Kermit, es gratuito). Una instalación de este tipo puede costar entre la cuarta y la tercera parte que el caso típico que estamos considerando, y eligiendo correctamente el software de los usuarios puede cubrir perfectamente todas las necesidades de la empresa, incluyendo edición profesional de textos. Por si fuera poco, la acción administrativa para dar de alta un terminal y configurarlo es trivial (véase más adelante en este tema), y una vez hecha no dará ninguna clase de problema en el futuro.

Más aún, parte del ahorro conseguido implantando terminales puede invertirse en mejorar las condiciones ergonómicas de los puestos, y en concreto, en dotarlos de teclados y monitores de calidad, que son a fin de cuentas los elementos más importantes.

Por último, no es despreciable el ahorro en gastos de administración y mantenimiento, que a lo largo del tiempo puede ser superior incluso al ahorro inicial en hardware.

4.2. Tipos de terminales

Existen tres tipos básicos de terminales: los que se comunican con el S.O. a través de un puerto de comunicaciones (serie o paralelo, aunque normalmente es el primero de ellos el elegido), los que lo hacen a través de la memoria y los que lo hacen a través de una interfaz de red. A su vez, los dos primeros admiten subdivisiones:

- Interfaz mapeada en memoria
  - orientada a carácter
  - orientada a bits
- Interfaz serie (normalmente)
  - tipo tty
  - tipo 'inteligente'

4.2.1. Terminales por interfaz de memoria

En estos terminales, una parte de la RAM, llamada 'RAM de vídeo' es leída por el controlador de vídeo. Consideremos los terminales por interfaz de RAM orientados a carácter, y en particular, centrémonos en la terminal tipo IBM PC. En el segmento 0xb000 en modo monocromo o 0xb800 en color, se encuentran los códigos ascii [1] de los caracteres que aparecerán en el terminal, junto con sus atributos. En una pantalla de 80 columnas por 25 líneas esto representa 4000 octetos, agrupados en 2000 parejas carácter-atributo. Cada carácter indexa una tabla que contiene mapas de bits de 8x16, cada uno de los cuales es la representación del carácter. De esta forma, después de leer los primeros 80 caracteres, se genera la primera línea horizontal de barrido del tubo de rayos catódicos. Siguen otros quince barridos y de esta forma queda completa la primera línea de texto en pantalla. La operación se repite con las 24 líneas restantes, completándose una pantalla alrededor de 70 veces por segundo. En los monitores a color, existen tres tubos independientes, cuyas intensidades se regulan por medio del octeto de atributos que acompaña a cada carácter. Finalmente, digamos que la tabla que contiene el mapa de bits para cada carácter se encuentra en ROM, pero la BIOS del IBM PC ofrece la posibilidad de usar cualquier otra tabla definida por el usuario.

En cuanto a los terminales mapeados orientados a bits, el funcionamiento es el mismo, solo que ahora cada octeto controla uno, varios o parte de un pixel en pantalla. Por ejemplo, usando un octeto por pixel puede asignarse a éste un valor de entre 256 distintos, que pueden ser colores o grises en una escala de 0 a 255.

4.2.2. Terminales por interfaz serie

Los terminales por interfaz serie se comunican con el S.O. a través de una línea serie, es decir, bit a bit. Normalmente se usa la interfaz RS-232, que se implementa con conectores de 9 o 25 pines. De éstos, uno se usa para recibir bits, otro para enviar, un tercero para tierra (referencia de voltaje) y el resto para distintas funciones de control, que rara vez se usan. Cuando se desea enviar un carácter, se envía un bit de inicio, seguido por los 7 u 8 bits del carácter, quizás un bit de paridad para efectuar una detección de errores rudimentaria y 1 o dos bits de parada. Estas transmisiones se realizan a velocidades típicas de entre 9600 y 38400 bps. En realidad, los terminales vía RS-232 como tales se encuentran en proceso de extinción,

[1] Más exactamente, lo que se encuentran son octetos que indexan la tabla que contiene los patrones de bits de los caracteres

pero no así la función que realizan. La razón es que están siendo sustituidos por ordenadores personales que emulan por software la terminal. Cuando el software que controla la terminal quiere enviar un carácter, lo escribe en la tarjeta de interfaz, de donde pasa a un chip especial, llamado UART, de complejo funcionamiento, pero que esencialmente se ocupa de la conversión del caracter a una secuencia de bits. Puesto que la comunicacion bit a bit es lenta, su gestión se realiza vía interrupción: se genera una interrupción cuando la UART ha terminado de enviar un carácter, y otra cuando acaba de recibirlo.

En su versión más simple, terminales tty (abreviatura de TeleType, empresa que los desarrolló hace muchos años), el carácter se envía a una impresora y aparece en papel (como dato curioso, digamos que la primera versión UNIX se escribió usando una terminal tty). Es raro ver hoy en día terminales tty basados en impresora, ya que hace mucho que fueron sustituidos por 'tty de cristal', que funcionan igual, solo que el carácter se representa en un monitor. A su vez, los terminales tty de cristal quedaron obsoletos hace tiempo.

En su versión *inteligente*, se dota al terminal de un procesador, un pequeño programa en ROM y algo de memoria para que sea capaz de realizar por sí solo ciertas funciones. Por ejemplo, puede interpretar secuencias de octetos para colocar el cursor en la posición que se desee, hacer desplazamientos, borrar selectivamente regiones de la pantalla y cosas por el estilo. Los terminales inteligentes continúan comercializándose para grandes equipos (bancos, centros de cálculo, bases de datos, etc) pero como se ha dicho antes, lo normal es usar un PC con un programa capaz de emular por software la función del terminal.

Llevados a sus últimas consecuencias, los terminales inteligentes cuentan con un potente procesador, varios MB de memoria RAM, teclado y ratón, junto con un monitor color de alta resolución, y se comunican con el anfitrión mediante una red de alta velocidad.

4.2.3. Terminales ANSI

De entre todos los terminales que se han llevado a la práctica, uno de los más extendidos es el terminal ANSI. Este terminal, o más exactamente, el protocolo mediante el cual se controla, se ha usado y se usa tanto en terminales clásicos vía serie como en terminales virtuales, tanto en modo texto como en modo gráfico. Se trata de un conjunto de órdenes que en los

terminales vía serie se ocupa de ejecutar el código en ROM en el propio terminal y en los terminales por interfaz de memoria el controlador software adecuado, intercalado de alguna forma entre los programas de aplicación y el controlador hardware.

Se trata de un terminal lo suficientemente extendido, y con un conjunto de órdenes lo suficientemente flexible como para que algunos [2] lenguajes de programación carezcan de librerías para el control de un terminal en modo texto de cursor direccionable y confíen en las capacidades del terminal ANSI que el sistema incorpore.

El control del terminal se efectúa de forma sencilla: un carácter especial, el carácter `ESC`, inicia una secuencia que es interpretada por el controlador software o por la ROM del terminal, y ejecuta, según la secuencia de que se trate, una u otra acción. Las secuencias de control pueden contener parámetros selectivos `Ps`, que no son sino números decimales, parámetros de línea, que indican la línea del terminal afectada y se representan por `PL` y parámetros de columna, que se indican mediante `Pc`. Estas son las secuencias más comunes:

`ESC[PL;PcH`	coloca el cursor en la posición especificada
`ESC[PnA`	mueve el cursor hacia arriba
`ESC[PnB`	mueve el cursor hacia abajo
`ESC[PnC`	mueve el cursor hacia adelante
`ESC[PnD`	mueve el cursor hacia atrás
`ESC[s`	guarda la posición del cursor
`ESC[u`	restaura posición del cursor
`ESC[2J`	borra la pantalla
`ESC[K`	borra desde la posición del cursor hasta el fin de la línea
`ESC[Ps;Ps...Psm`	establece el modo gráfico, según tabla siguiente

[2] ¿o muchos?

0	desactiva todos los atributos
1	resaltado
4	subrayado
5	parpadeo
7	video inverso
30	fondo negro
31	fondo rojo
32	fondo verde
33	fondo amarillo
34	fondo azul
35	fondo magenta
36	fondo cián
37	fondo blanco
40	texto negro
41	texto rojo
42	texto verde
43	texto amarillo
44	texto azul
45	texto magenta
46	texto cián
47	texto blanco

4.3. Modos de entrada

Consideremos lo que ocurre cuando un proceso efectúa una petición de entrada de carácter vía teclado. Puesto que el tiempo de respuesta del usuario es órdenes de magnitud superior al tiempo de procesamiento de la CPU, el proceso peticionario queda bloqueado, en espera de respuesta. Cuando ésta se produce, la tarea que controla la interrupción del teclado toma el código de la tecla pulsada del puerto correspondiente y lo pone en un *buffer*, quizás después de haber efectuado una conversión. El S.O. pondrá al proceso peticionario en estado de *listo para ejecutarse* y en su siguiente turno podrá tomar el carácter que la tarea de terminal había depositado en el *buffer*.

Pero, ¿que ocurre con la pantalla? Existen aquí dos enfoques diferentes posibles. O bien la misma tarea que lee del teclado es la encargada de representar el carácter en pantalla, o bien son dos tareas distintas.

Otro aspecto a tener en cuenta es si el controlador de teclado efectúa algún procesamiento antes de pasar datos al proceso peticionario. Por ejemplo,

los editores de pantalla avanzados como **vim** quieren un flujo de octetos tal y como son tecleados por el usuario. De esta manera, pueden conectar cualquier pulsación con cualquier acción. La otra posibilidad es que al proceso peticionario sólo le interesen líneas completas, y no la forma en que se crearon, ya que pudieron cometerse errores y el usuario pudo borrar y volver a escribir algunos caracteres. A esta segunda posibilidad, la norma POSIX la llama *modo canónico*. A la primera, *modo no canónico*.

4.4. La base de datos y librería termcap

4.4.1. Introducción

termcap es una librería normalizada en UNIX que permite a los programas usar terminales de forma transparente. La librería y base de datos asociada es originaria de Berkeley UNIX.

La base de datos describe las capacidades de cientos de terminales con alto grado de detalle. Por ejemplo, las dimensiones en filas y columnas, qué orden es preciso enviar para mover el cursor, hacer desplazamientos de pantalla, etc.

Por su parte, la librería provee de funciones para acceder a la base de datos y extraer la información. Podemos decir que la librería es la API para la base de datos, y contiene funciones para:

- encontrar la descripción de un determinado tipo de terminal.
- extraer de dicha descripción información particular
- calcular y realizar envíos de relleno (explicación más adelante)
- codificar parámetros numéricos (p. ej. fila y columna donde mover el cursor) en la forma requerida por el terminal

4.4.2. Preparación

Para usar la librería *termcap* en un programa, existen dos tipos de requisitos:

- el compilador necesita declaraciones de las funciones y variables de la librería. En los sistemas GNU, es suficiente incluir el archivo de cabecera *termcap.h*

- el enlazador necesita encontrar la librería. Usualmente, es suficiente con añadir el argumento `-ltermcap` o `-ltermlib` al llamar al compilador

```
gcc miprograma -ltermcap
```

4.4.3. Búsqueda de una descripción de terminal

Un programa de aplicación que va a usar *termcap* debe primero buscar la descripción del terminal que va a usar. Esto se hace a traves de la funcion `tgetent()` (-t-erminal -get- -ent-ry), cuya declaración es:

```
int tgetent(char *BUFFER, char *TERMTYPE)
```

Esta función busca la entrada y la almacena internamente, de manera que a continuación pueda extraerse de ella la información que se desee. El argumento `TERMTYPE` es una cadena con el nombre del terminal que se está buscando. Usualmente, el programa de aplicación lo obtiene de la variable de entorno `TERM` usando `getenv("TERM")`.

Por su parte, `BUFFER` es lo que parece: espacio para guardar la información. En la mayoría de los UNIX es preciso reservar manualmente el espacio que se necesite, pero en las versiones GNU, pasando en `BUFFER` un puntero NULL es *termcap* quien se encarga de hacer la reserva usando `malloc()`.

En cualquier caso, *termcap* guarda internamente la dirección del *buffer* para usarla posteriormente cuando mediante `tgetnum()`, `tgetflag()` o `tgetstr()` se le pregunte por aspectos específicos. Por otra parte, una segunda llamada a `tgetent()` libera el espacio reservado la primera vez.

En cuanto al valor de retorno, éste es -1 si no pudo accederse a la base de datos, 0 si pudo accederse pero no se encontró el terminal especificado, y otro número en cualquier otro caso. Véase el siguiente código de ejemplo:

```
#ifdef UNIX
static char term_buffer[2048]
#else
#define term_buffer 0
#endif
init_terminal_data()
{
    char termtype=getenv("TERM");
```

```
    int bien;

    if (termtype==0)
        printf("error:especificar tipo de terminal con setenv\n");

    bien=tgetenv(term_buffer,termtype);

    if (bien<0)
        printf("error: no se pudo acceder a termcap\n");
    else
    if (bien==0)
        printf("error: no existe terminal\n");
}
```

4.4.4. Extrayendo información del buffer

Cada elemento de información contenido en el *buffer* es lo que se llama una 'capacidad' del terminal, y cada una de estas capacidades tiene un nombre de dos letras. Por ejemplo 'co' es el número de columnas que soporta el terminal. Algunas de las capacidades, como 'co', son numéricas; otras son booleanas (la capacidad esta disponible o no) y otras son cadenas. Cada capacidad es siempre del mismo tipo. Por ejemplo, 'co' es siempre un número, 'am' es siempre un booleano (automatic wrap margin) y 'cm' (cursor motion) es siempre una cadena. En consecuencia, según el tipo de capacidad se usará una de las siguientes funciones:

```
int tgetnum(char *NAME)
int tgetflg(char *NAME)
int tgetstr(char *NAME, char **AREA)
```

Tanto `tgetnum()` como `tgetflg()` toman un sólo argumento, que es el nombre de la capacidad por la que se está preguntando. `tgetnum()` devuelve un número que es el valor de la capacidad, o -1 si dicha capacidad no se menciona en la base de datos. `tgetflg()` devuelve 1 o 0 según que la capacidad esté presente o no. En cuanto a `tgetstr()`, ha de pasársele el nombre de la capacidad y un puntero a una cadena, donde se copiará el resultado. ¿Qué tamaño ha de tener la cadena? En las versiones GNU, pasando un puntero NULL en `AREA`, `tgetstr()` usará `malloc()` para reservar el espacio necesario. Una vez hecho esto, termcap no lo usará nunca más, ni se referirá a él, por lo que una vez leído puede liberarse el espacio.

Considérese el siguiente fragmento de código:

```
char *tgetstr ();
char *cl_string, *cm_string;
int height;
int width;
int auto_wrap;

char PC;   /* For tputs.  */
char *BC;  /* For tgoto.  */
char *UP;

interrogate_terminal ()
{
    #ifdef UNIX
    /* Here we assume that an explicit term_buffer
      was provided to tgetent.  */
    char *buffer = (char *) malloc (strlen (term_buffer));
    #define BUFFADDR &buffer
    #else
    #define BUFFADDR 0
    #endif

    char *temp;

    /* Extract information we will use.  */
    cl_string = tgetstr ("cl", BUFFADDR);
    cm_string = tgetstr ("cm", BUFFADDR);
    auto_wrap = tgetflag ("am");
    height = tgetnum ("li");
    width = tgetnum ("co");

    /* Extract information that termcap functions use.  */
    temp = tgetstr ("pc", BUFFADDR);
    PC = temp ? *temp : 0;
    BC = tgetstr ("le", BUFFADDR);
    UP = tgetstr ("up", BUFFADDR);
}
```

4.4.5. Preparación

Antes de comenzar a enviar información al terminal, bien para que muestre caracteres, bien para que ejecute órdenes, es preciso hacer un par de tareas previas:

- hay que iniciar algunas variables públicas, usadas por *termcap*, como `PC`, `ospedd`, `UP` y `BC`
- instruir al *kernel* para que no efectúe transformaciones sobre los tabulados horizontales. Esto sólo es preciso con terminales muy antiguos, y requiere además que el programa capture las llamadas `SIGQUIT` y `SIGINT`. Nosotros no lo tendremos en cuenta.

4.4.6. Envíos de relleno

En determinadas circunstancias, es preciso enviar una secuencia de caracteres nulos al terminal. Ocurre que muchos terminales tienen órdenes que invierten mucho en ejecutarse. Por ejemplo, limpiar la pantalla puede llevar 20 ms, y en ese tiempo, por una línea serie a 9600 bps pueden llegar veinte caracteres, que se perderían al estar ocupado el terminal en otra tarea. Aunque todos los terminales tienen un *buffer*, una secuencia de órdenes lentas puede llenarlo. La solución consiste en enviar caracteres nulos, de forma que no puedan perderse caracteres útiles. El número de caracteres de relleno depende de la velocidad de la línea, claro está.

A veces se observa un comportamiento extraño en un terminal, debido a que el *buffer* se encuentra lleno y a que alguna orden ha sido desechada. Este comportamiento normalmente está asociado a las órdenes que hacen desplazamientos y limpian la pantalla.

En la base de datos, el tiempo durante el que se precisa enviar caracteres nulos aparece como un número en milisegundos, quizás con parte decimal. Además, puede ser preciso esperar más o menos tiempo para la misma orden según el número de líneas afectadas. Por ejemplo, no es lo mismo insertar una línea al principio de la pantalla que cerca de su parte inferior. El carácter '*' siguiendo al tiempo de relleno significa que este tiempo debe ser multiplicado por el número de líneas afectadas. Por ejemplo:

```
:al=1.3*\E[L
```

sirve para describir la inserción de una línea en cierto terminal. La orden en sí es `ESC [ L`, y el tiempo de relleno de 1.3 ms por el número de líneas afectadas.

Finalmente, existen dos variables globales que es preciso conocer. `pc` indica qué carácter va a ser enviado como relleno. Normalmente, sera NULL (0). `pb` indica por debajo de qué velocidad no es necesario el relleno.

4.4.7. Forma de especificar los parámetros

Algunos controles al terminal requieren parámetros numéricos. Por ejemplo, al mover el cursor es preciso especificar la fila y la columna. El valor de la capacidad `cm` (cursor movement) no puede ser simplemente una tira de caracteres. Debemos saber cómo expresar los números que dan la posición del cursor y qué lugar ocupan en la cadena. Las especificaciones de termcap incluyen convenciones para saber si una cadena tiene o no parámetros, cuantos y qué significado tienen. Por ejemplo, para `cm` se requieren dos posiciones, una horizontal y otra vertical, siendo (0,0) el valor de la esquina superior izquierda.

termcap también define un lenguaje, usado en la definición de la capacidad, para especificar cómo y dónde se codifican los parámetros. Este lenguaje usa secuencias de caracteres que comienzan por '%'.

Un programa que está haciendo salida a pantalla llama a las funciones `tparam()` o `tgoto()` para codificar los parámetros de acuerdo con las características del terminal. Estas funciones producen una cadena con la orden que se enviará al terminal.

4.4.8. El lenguaje de codificación de parámetros

Como se ha dicho, una orden de cadena que requiere parámetros contiene secuencias especiales de caracteres comenzando con '%' para expresar como han de codificarse los parámetros. Estas secuencias son interpretadas por `tparam()` y `tgoto()` para generar las órdenes. Los valores de los parámetros que se pasan a alguna de estas funciones se consideran formando un vector. Un puntero en este vector determina el siguiente parámetro a ser procesado. Algunas de las secuencias codifican un parámetro y avanzan el puntero al siguiente parámetro, otras alteran el puntero o el parámetro sin generar salida alguna.

Por decirlo de manera sencilla, lo que la base de datos *termcap* contiene son mini-programas en un lenguaje especial que `tparam()` y `tgoto()` interpretan para generar la cadena que se enviará al terminal. Por ejemplo, la cadena 'cm' para el terminal ANSI se escribe:

```
\E[%i%d;%dH
```

Donde
`E` es el caracter ESC. 'cm', por convención, siempre requiere dos parámetros: fila y columna, y por eso la cadena anterior especifica la codificación de dos parametros. `%i` incrementa los dos valores que se proporcionarán a `tparam()` o `tgoto()` en la llamada. Cada `%d` codifica uno de ellos en decimal. Si la posición del cursor deseada es (20,58), el resultado que se envía al terminal es:

```
\E[21;59H
```

Las 'secuencias-%' que generan salida son las siguientes. Excepto '%%', el resto codifican un parámetro y avanzan el puntero una posición.

- **%%** es la única forma de representar un '%' literalmente dentro de una orden que lo precise.
- **%d** codifica el siguiente parámetro en decimal
- **%2** igual, solo que usa siempre dos dígitos, al menos
- **%3** igual que el anterior, pero usando siempre tres dígitos, al menos. %x con x mayor que 3 no está definido.
- **%.** toma el siguiente parámetro como un carácter ASCII simple cuyo código es el valor del parámetro. Es similar al %c del `printf()` de C.
- **%+CHAR** suma el siguiente parámetro al carácter CHAR, y devuelve el carácter resultante. Por ejemplo '%+ ' representa 0 como espacio, 1 como '!', etc.

Las siguientes secuencias-% alteran el orden o el valor de los parámetros. No generan salida:

- **%i** incrementa los siguientes parámetros
- **%r** intercambia los siguientes dos parámetros. Se usa en terminales que esperan en primer lugar el número de columna.

- **%s** salta al siguiente parámetro: no hace nada
- **%b** retrocede un parámetro
- **%C1C2** incrementa condicionalmente el siguiente parámetro. C1 y C2 son caracteres ASCII dados por sus números correspondientes. Si el siguiente parámetro es mayor que C1, se le suma el código ASCII de C2.

Estos son los más importantes, pero hay otros, especialmente las extensiones GNU y órdenes especiales creadas *ad hoc* para ayudar a describir terminales especialmente esotéricos.

4.4.9. Enviar órdenes con parámetros

Las funciones `tparam()` y `tgoto()` son los equivalentes a `printf()` para enviar órdenes al terminal. El primero es una extensión GNU, mientras que el segundo es genuinamente UNIX, y es preferible para mover el cursor.

La función tparam()

Esta función puede codificar órdenes de terminal con cualquier número de parámetros y permite especificar el tamaño del *buffer* para contenerlos. Es la función preferida para codificar cualquier capacidad excepto 'cm'. Su declaración en ANSI C es la siguiente:

```
char * tparam(char *CTLSTRING, char *BUFFER,
              int SIZE, int PARAM1, ...)
```

Los argumentos son una cadena de control (presumiblemente, el nombre de una capacidad del terminal), un *buffer* de salida, su tamaño y cualquier número de parámetros que han de codificarse. El efecto de la función es codificar la orden y dejarla en el *buffer* . Si su tamaño no es suficiente, llama a `malloc()` para procurar más espacio. Todas las capacidades que requieren parámetros pueden necesitar relleno, y por eso ha de usarse la función `tputs()` para enviar la cadena al terminal.

La función tgoto()

`tgoto()` es la única función disponible en UNIX, y maneja bien problemas que pueden darse con caracteres nulos, tabuladores y caracteres nueva línea en ciertos terminales, por lo que es el método preferido para posicionar el cursor. Su declaración es:

```
char *tgoto(char *CSTRING, int HPOS, int VPOS)
```

La función compone la cadena de control y la almacena en un *buffer* interno, devolviendo la dirección de dicho *buffer* .

Los caracteres `NULL`, `TAB` y nueva línea pueden crear problemas. El primero, porque `tputs()` puede creer que ha llegado al final de la cadena. El segundo, porque el núcleo u otro programa puede sustituirlo por espacios y el tercero porque el núcleo puede añadirle un retorno de carro. Para evitar problemas, `tgoto()` incrementa en uno los caracteres (0,9,10) ,añadiendo después una orden para ir un espacio arriba o atrás. Estas cadenas de compensación se encuentran en las variables globales `BC` y `UP`, y es responsabilidad del programador poner en ellas los valores correctos, normalmente obtenidos de 'le' y 'up' con `tgetstr()`.

Formato de la base de datos

Excepto las líneas que comienzan con '#', que son comentarios, y las que se encuentran en blanco, cada línea de *termcap* es una descripción de un terminal. Ocurre que, por diseño, cada descripción de terminal ocupa una línea lógica, pero por conveniencia a la hora de editar el archivo, cada línea lógica puede descomponerse en una serie de líneas 'físicas'.

He aquí una entrada real del archivo `/etc/termcap`:

```
vt52|dec vt52:\
:co#80:it#8:li#24:\
:bl=^G:cd=\EJ:ce=\EK:cl=\EH\EJ:cm=\EY%+ %+ :cr=^M:\
:do=\EB:ho=\EH:kb=^H:kd=\EB:kl=\ED:kr=\EC:ku=\EA:\
:le=\ED:nd=\EC:nw=^M^J:sf=^J:sr=\EI:ta=^I:up=\EA:
```

Cada descripción de terminal comienza con algunos nombres para el tipo de terminal. Estos nombres se encuentran separados por '|', y se coloca un ':' a continuación del último nombre. Por compatibilidad con UNIX antiguos, el primer nombre tiene sólo dos caracteres. El último puede ser una frase completa, como `dec vt52` o algo como `terminal minix conforme POSIX`. Los otros nombres pueden ser cualquier cosa que el usuario pudiese teclear para especificar su terminal.

Después del tipo de terminal, vienen las capacidades, separadas por ':', y con ':' después de la última. Cada capacidad tiene un nombre de dos letras. Por ejemplo, 'cm' por 'cursor motion' y 'li' por 'number of lines'.

Como se dijo arriba, existen tres clases de capacidades: banderas, números y cadenas. Cada clase se escribe de una forma en la descripción. Una función particular siempre se llama con el mismo nombre, y es siempre de la misma clase. Por ejemplo, 'cm' siempre se llama 'cm', y siempre es una cadena.

Las capacidades tipo bandera pueden ser verdaderas o falsas. Si el nombre está, es verdadera, si no está, es falsa.

Las capacidades numéricas son enteros positivos. Se escribe la capacidad, el caracter '#' y el número. Por ejemplo, para decir que un terminal tiene 48 líneas habrá de aparecer `:li#48:`.

Una capacidad cuyo valor es una cadena se define con su nombre, un '=' y la cadena. Por ejemplo, los terminales ANSI tiene la siguiente capacidad para mover el cursor:

```
:cm=\E[%i%d;%dH
```

El carácter `ESC` se representa por `\E`. Un carácter de control se representa mediante un acento circunflejo, y a su vez la barra *backslash* y el circunflejo se representan colocando un segundo *backslash* delante.

La cadena puede contener números al inicio para especificar rellenos y/o secuencias- % para codificar los parámetros. Volviendo a las especificaciones para el vt52, encontramos dos capacidades numericas, 'co' y 'li', dos banderas, 'bj' y 'pt' y varias cadenas, la mayoria de las cuales comienzan con ESC.

Conclusión

Esta sección podría ser mucho mas larga, pero creo que es suficiente este resumen de las páginas 'info' GNU sobre *termcap*. Allí se encuentra toda la información adicional sobre capacidades en concreto y otros aspectos avanzados que no hemos incluido. Sólo restan dos observaciones. La primera, que durante mucho tiempo se usaron terminales basadas en impresoras de líneas, y que por tanto `termcap` (por ejemplo, en AIX) contiene descripciones de las órdenes de control de impresora difíciles de encontrar por otros medios.

Segundo, que existe un sistema alternativo más moderno para descripción de terminales llamado *terminfo*. Aquí, las descripciones se encuentran en una serie de directorios ordenados alfabéticamente, de forma que, por ejemplo, el terminal ANSI se encontraría en /A, el vt52 en /V, etc. Además, la descripción de un terminal en concreto no es un archivo ASCII, sino que se encuentra compilada en un formato distinto. La herramienta para compilar descripciones de terminal se llama `tic`, y remito a las páginas 'man' o 'info' para ampliar la información. He preferido *termcap* por su formato ASCII fácil de consultar, ampliar, modificar y portar.

Tres lecciones adicionales se pueden obtener de toda esta exposición.

Primera: en una época en que la instalación de nuevo hardware requiere a su vez la instalación de controladores en formato binario suministrados por el fabricante y gestionados por el Sistema Operativo, es interesante ver que existen soluciones mucho más sencillas y elegantes. Un fabricante de terminales, sólo necesitaba facilitar un pequeño archivo de texto, de menos de diez líneas, para conseguir que cualquier programa, directamente, pudiese ser usado sobre su hardware.

Segunda: la interfaz universal entre programas es una corriente de texto. Lo vemos en la base de datos `termcap`. Además, esta corriente de texto es directamente legible por el programador o usuario, facilitando enormemente la resolución de problemas o depuración de programas cuando se producen errores o comportamientos no esperados.

Tercera: un pequeño lenguaje de programación es una herramienta fácil de construir y tremendamente útil. Si se mira con atención, descubrimos muchos micro-lenguajes en sitios donde no imaginábamos. Por ejemplo, la cadena de formato de la función `printf()`, o las pequeñas cadenas que definen las capacidades de los terminales en la base de datos `termcap`.

4.5. La librería curses.h

4.5.1. Introducción

Como se ha dicho anteriormente, la librería *termcap* permite programar interfaces orientadas a terminal de una forma independiente del terminal. La librería *curses.h* persigue el mismo objetivo, pero a un nivel ligeramente más alto. De hecho, *curses* se apoya en *termcap*, pero, a diferencia de este último:

- *curses.h* se encuentra portada a todos los sistemas operativos: DOS y sucesores, todos los UNIX, OS/2, Amiga...
- en aquellos sistemas que no disponen de *termcap*, *curses* maneja directamente el terminal.
- *curses* implementa un sistema de ventanas, y gestiona las superposiciones, desplazamientos y actualizaciones. Además, realiza las optimizaciones necesarias para minimizar el flujo de octetos entre el anfitrión y el terminal, que recordemos puede estar unido al primero por una lenta conexión serie.

4.5.2. Funciones básicas

La librería *curses.h* provee de gran cantidad de funciones para definir ventanas, colocar texto en posiciones específicas, cambiar los atributos y colores y aceptar entradas desde teclado, incluyendo teclas de función. Para que todas estas funciones sean accesibles es preciso iniciar el entorno de ventanas, antes de la llamada a cualquiera de estas funciones, así como cerrarlo una vez concluido el programa. De esta forma, el programa mínimo que podría hacer uso de la librería es:

```
#include <curses.h>
main()
{
    initscr();
    endwin();
}
```

`initscr()` inicia el entorno de ventanas abriendo una ventana por defecto cuyo tamaño coincide con el de la pantalla en modo texto con 80 columnas y 25 filas, y `endwin()` finaliza el entorno de ventanas. Al inicio del código suelen añadirse tras `initscr()` dos funciones que permiten eliminar el eco para las entradas desde teclado e interrumpir los programas con `CTRL+C`, si fuese preciso:

```
#include <curses.h>
main()
{
    initscr();
    cbreak();
    noecho();
```

```
    /* codigo del programa */
    endwin();
}
```

El tipo fundamental que usan este tipo de programas es `WINDOW`, definido en *curses.h*. Este tipo es una estructura que contiene información sobre el número de filas y columnas, posición de la ventana sobre la pantalla, posición del cursor en la ventana, atributos, etc. Como la mayoría de las veces los programas se ejecutarán sobre terminales, la librería incluye código para minimizar el número de octetos que han de moverse cuando se escribe o redibuja una ventana. Obsérvese el código siguiente:

```
#include <curses.h>
main()
{
    WINDOW *miventana;
    initscr();
    cbreak();
    noecho();
    miventana=newwin(0,0,0,0);
    wrefresh(miventana);
    /* codigo del programa */
    delwin(miventana);
    endwin();
}
```

La función `newwin()` es la encargada de iniciar la variable de tipo `WINDOW *miventana`. Esta función toma como parámetros el número de filas, el número de columnas, la primera fila y la primera columna de nuestra ventana, cantidades referidas a la pantalla 80 por 25 (en principio). Cuando se pasan todos los parámetros a cero, se crea una ventana del tamaño del terminal. Una vez creada la ventana, podemos, usando las funciones específicas, escribir en posiciones concretas con los atributos deseados, pero la actualización de la ventana no será automática, sino que habrá que forzarla mediante la funcion `wrefresh()`, que toma como argumento una variable tipo `WINDOW *`. Para finalizar, liberaremos la memoria usada por las ventanas que hayamos definido y daremos por finalizada la sesión de ventanas mediante `endwin()`.

4.5.3. Funciones comunes

Existen dos grandes familias de funciones en la librería *curses.h*. Por un lado se encuentran las funciones que comienzan por, o contienen a, la letra 'w' (de Window). El resto no hacen referencia a ninguna ventana en concreto definida en el programa, sino a la ventana por defecto, `stdscr`, iniciada al cargar la librería. De las primeras, las más comunes permiten situar el cursor en una posición determinada de la pantalla y escribir caracteres simples o cadenas de texto con atributos específicos. Las más usadas, como decimos, son las siguientes:

```
move(y,x)
wmove(win,y,x)
```
El cursor asociado con la ventana por defecto, o con la ventana `win` es colocado en la fila `y` columna `x`.

```
addstr(str)
waddstr(win,str)
mvaddstr(y,x,str)
mvwaddstr(win,y,x,str)
```
Las dos primeras imprimen en la ventana por defecto, o en la ventana `win`, la cadena `str`, que ha de terminar con el caracter `NULL`. La impresión se efectúa a partir de la posición actual del cursor. La tercera y cuarta colocan previamente el cursor en la posición deseada, bien de la ventana por defecto, bien de `win`, y entonces imprimen la cadena `str`.

```
addch(ch)
waddch(ch)
mvaddch(y,x,ch)
mvwaddch(win,y,x,ch)
```
Es la familia de funciones totalmente análoga a la anterior, pero operando con caracteres individuales en lugar de con cadenas. En realidad, las funciones del grupo anterior son macros que se apoyan en este grupo.

```
clear()
wclear(win)
clrtobot()
wclrtobot(win)
clrtoeol()
wclrtoeol(win)
```
La primera borra la ventana por defecto, y la segunda la ventana `win`. Ténga-

se no obstante en cuenta que el borrado efectivo sólo se llevará a cabo en la siguiente llamada a `refresh()` o `wrefresh()`. Las funciones tercera y cuarta borran desde la línea en que se encuentra el cursor, incluida, hasta la última línea de la ventana. Finalmente, las funciones quinta y sexta borran desde el cursor, incluido, hasta el final de la línea actual.

```
delch()
wdelch(win)
mvdelch(y,x)
mvwdelch(win,y,x)
```

Emparentadas con las funciones de la familia anterior, éstas permiten borrar el carácter de la ventana actual, o de la ventana por defecto, que se encuentra en la posición del cursor, o en una posición específica.

```
printw(fmt,args)
wprintw(win,fmt,args)
mvprintw(y,x,fmt,args)
mvwprintfw(win,y,x,fmt,args)
```

Para finalizar con el conjunto de funciones de salida a pantalla, esta familia es el análogo a la función `printf()` de C, y permite salidas con formato. El formato tiene la misma sintaxis que en `printf()` y los argumentos `args` son impresos en la posición del cursor en la ventana por defecto o en la ventana actual, o bien en una fila y columna dados.

```
getch()
wgetch(win)
mvgetch(y,x)
mvwgetch(win,y,x)
```

Este es el conjunto básico de funciones de entrada. `getch()` toma un carácter del teclado, devolviendo un entero. Si se ha llamado previamente a la función `noecho()`, el carácter en cuestión no es impreso en pantalla. Es el modo preferido para los programas interactivos. En principio, `getch()` devuelve un único entero sólo para las teclas alfanuméricas, pero normalmente estaremos interesados en capturar las teclas de función y las flechas arriba, abajo, izquierda y derecha. Esto se consigue haciendo una llamada previa a la función `keypad(win,TRUE)`, llamada que normalmente se hace junto con `initscr()` o `newwin()`. De esta forma, cuando se pulse una tecla de función, no se devuelve una secuencia de números, sino un *token* para esa tecla de función. Los valores posibles se encuentra en *curses.h*, y son enteros que comienzan con 0401, mientras que los nombres de las teclas comienzan con `KEY_`. Cuando se recibe un valor que puede ser el comienzo de una tecla de función, *curses*

inicia un temporizador. Si este vence, se devuelve la tecla pulsada, y si no se devuelve la tecla de función. El valor máximo que ha de alcanzar el temporizador depende de cada terminal, y por ese motivo en algunos terminales se produce un cierto retraso entre la pulsación de la tecla `ESC` y la respuesta a la misma por parte del programa. El listado completo de las teclas definidas en *curses* puede encontrarse en el manual de referencia de UNIX System V/386, de AT&T, pero las más comunes son las siguientes:

```
KEY_BREAK        0401
KEY_DOWN         0402
KEY_UP           0403
KEY_LEFT         0404
KEY_RIGHT        0405
KEY_HOME         0406
KEY_BACKSPACE    0407
KEY_F0           0410
KEY_F(n)         KEY_F0+(n)
```

4.5.4. Atributos y colores

Nos referiremos en primer lugar a aquellos atributos que no involucran colores. Cuando desea imprimirse en pantalla con atributos, el atributo específico ha de activarse con la función `wattron(ATRIBUTO)`. Cualquier cosa que se imprima a continuación lo hará con el atributo definido por `wattron()`, por lo que usualmente se usa la pareja `wattron()`, `wattroff()`. La primera lo activa, y la segunda lo desactiva. Entre ambas se coloca el código para imprimir con el atributo deseado. Los atributos se nombran por sus macros, definidas en *curses*. Los mas comunes son:

```
A_UNDERLINE      subrayado
A_REVERSE        video inverso
A_BLINK          parpadeo
A_BOLD           resaltado
```

El siguiente programa define una ventana y escribe "LINUX" en vídeo inverso en la posición 10,10:

```
#include <curses.h>
main()
{
    WINDOW *miventana;
```

```
    initscr();
    miventana=newwin(0,0,0,0);
    wattron(miventana,A_REVERSE);
    mvwaddstr(miventana,10,10,"LINUX");
    wattroff(miventana,A_REVERSE);
    wrefresh(miventana);
    wgetch(miventana);
    /* termino */
    delwin(miventana);
    endwin();
}
```

Consideremos ahora el uso de los colores. Para que un programa pueda hacer uso de las funciones que tratan con los colores, es preciso antes que nada llamar a la rutina `start_color()`, que no necesita argumentos. Como `start_color()` ha de llamarse antes que cualquier otra rutina de manipulación del color, es una buena práctica hacerlo inmediatamente después de `initscr()`. `start_color()` inicia ocho colores básicos, a saber: negro, azul, verde, cyan, rojo, magenta, amarillo y blanco. También inicia dos variables públicas llamadas `COLORS` y `COLOR_PAIRS` que definen respectivamente el máximo número de colores y de parejas de colores (texto y fondo) que puede soportar el terminal. Finalmente devuelve `ERR` si el terminal no ofrece colores, y `OK` en caso contrario. A las rutinas que toman un color como argumento se puede pasar el número del color, o una de las macros definidas en *curses*:

```
COLOR_BLACK     0
COLOR_BLUE      1
COLOR_GREEN     2
COLOR_CYAN      3
COLOR_RED       4
COLOR_MAGENTA   5
COLOR_YELLOW    6
COLOR_WHITE     7
```

La función complementaria de `start_color()` es `has_colors()`, que también se llama sin argumentos y devuelve `TRUE` si el terminal permite manipulación de colores y `FALSE` en caso contrario. Esta función está indicada para escribir programas independiente del terminal. Es decir, el programador decide mediante la llamada a la función en el momento adecuado si va a usar colores o por el contrario se conformará con alguno de los atributos normales, como subrayado o vídeo inverso.

La rutina básica para la manipulación de los colores es `init_color()`. Esta rutina cambia la definición de un color, y toma cuatro argumentos: el color que se desea cambiar y las tres componentes de rojo, verde y azul. El primer argumento es un valor entre 0 y `COLORS-1`, y las componentes son tres valores entre 0 y 1000. Cuando se llama a `init_color()` cualquier ocurrencia del color que se redefina en la pantalla se actualiza inmediatamente a los nuevos valores. A la inversa, la funcion `color_content(color,&r,&g,&b)` permite averiguar las componente de un color determinado. Los colores no se usan aisladamente, sino en parejas (fondo,texto). Ya hemos dicho que `start_color()` inicia una variable pública llamada `COLOR_PAIRS` que informa del número de parejas disponibles. Existen funciones para averiguar qué colores componen cada pareja, y para cambiarlos a voluntad. Así, `init_pair(pair,f,b)` toma como argumentos el número del par que se desea cambiar, el número del color para el texto y el número del color para el fondo. A la inversa, `pair_content(pair,&f,&b)` devuelve, para un par determinado, los colores del texto y el fondo. Un pequeño programa nos permitirá aclarar las llamadas a las distintas funciones. En él, imprimiremos en color amarillo sobre fondo azul la frase `UNIX en COLORES`.

```
#include <curses.h>
main()
{
    WINDOW *miventana;
    initscr();
    cbreak();
    noecho();
    miventana=newwin(0,0,0,0);
    if (start_color()==OK){
        init_pair(2,COLOR_YELLOW,COLOR_BLUE);
        wattron(miventana,COLOR_PAIR(2));
        mvwaddstr(miventana,10,10,"UNIX en COLORES");
        wattroff(miventana,COLOR_PAIR(2));
        wrefresh(miventana);
    } else
    {
        mvwaddstr(miventana,10,10,"COLORES NO DISPONIBLES");
    }
    wgetch(miventana);
    delwin(miventana);
    endwin();
}
```

4.5.5. Caracteres gráficos

curses contiene una serie de caracteres gráficos predefinidos que permiten dibujar recuadros, flechas, rebordes, etc. Pueden usarse con la familia de funciones `addch()` y, en caso de que el terminal no los soporte, están previstos caracteres por defecto que los sustituirán.

4.5.6. Conexión y puesta en marcha de una terminal bajo Linux

La puesta en marcha de un terminal conectado a Linux es trivial. Normalmente, no se dispone de un terminal como tal, sino de un PC que corre un programa de emulación de terminal, como Kermit o Telix bajo DOS. No trataremos aquí el aspecto hardware del problema, limitándonos a decir que casi siempre el terminal se conecta vía serie, usando un cable NULL-modem.

Una vez conectadas las dos máquinas vía serie, usando `/dev/ttySx` en Linux y `COMy` en la máquina que suponemos ejecuta DOS, donde 'x' e 'y' son los números del puerto serie, editaremos como superusuarios el archivo `/etc/inittab`. Allí encontraremos comentadas algunas líneas que sirven para iniciar un proceso que se ocupa del puerto elegido. Una de estas líneas puede tener el aspecto siguiente:

```
s2:45:respawn:/sbin/agetty -L 9600 ttyS1
```

Además, de 'agetty' disponemos de otros programas, como 'getty' o 'ugetty'. Consúltese el manual. El parametro `-L` indica que la línea es local, y por tanto, que no es preciso comprobar la existencia de portadora. A continuación se indica la velocidad. Una opción conservadora son los 9600 baudios. Finalmente, se indica el dispositivo físico por donde se produce la comunicación. En este caso, `/dev/ttyS1`.

Este es un ejemplo, pero la línea anterior puede variar entre distribuciones. En mi máquina es la siguiente:

```
T1:23:respawn:/sbin/getty -L ttyS1 9600 vt220
```

donde se indica que el terminal es del tipo vt220.

Por su parte, en la máquina DOS sólo es preciso iniciar un programa de emulación de terminal. Como existen varios, es preciso seguir las instrucciones propias de cada uno de ellos. 'kermit' es un programa de emulación de terminal gratuito y potente, por lo que lo recomiendo. Básicamente, tanto con 'kermit' como con otro, habremos de comprobar que coincidan las velocidades seleccionadas en las dos máquinas, el número de bits por octeto y la paridad. También la emulación. Si se cumplen todas las condiciones, veremos que en la pantalla del PC ejecutando DOS aparece una línea familiar:

```
login:
```

Ya tenemos acceso a la máquina Linux. Suele descuidarse en este punto un importante aspecto. ¿ Dispondrá el usuario de todo el software que precise en modo carácter ?. Sin duda. Recordemos que la justificación del sistema UNIX fue la creación de herramientas de proceso de texto para la compañia AT&T. Existen procesadores de texto potentes, profesionales y eficientes, como troff, la versión groff o LaTeX. También existen en modo carácter procesadores interactivos como WordPerfect. No voy a listar aquí todo el software disponible para UNIX/Linux, basta con indicar que éste existe y que puede satisfacer cualquier necesidad de edición de texto, consultas y gestión de bases de datos, cálculo, desarrollo de programas, comunicaciones y un largo etc.
[3]

[3]En cualquier caso, tampoco es difícil configurar un terminal gráfico. Remitimos al artículo "Sesiones Remotas", aparecido en "sólo Linux", número 21, página 12, escrito por Jose Luis González.

Capítulo 5

Interfaz serie

5.1. Introducción

La transmisión de información vía serie es una forma natural de comunicar datos sobre líneas simples, como las antiguas líneas telegráficas, que sólo podían alternar entre dos estados distintos. De esta forma, a uno de los estados se le puede asignar un '0', mientras que al otro se le asigna un '1'. Cualquier sistema de codificación binario puede por tanto hacer uso de una línea serie simple como la descrita. No es de extrañar que la mayor parte de las computadoras dispongan de uno o varios de estos interfaces, que a su vez pueden implementarse de varias formas. Por ejemplo, la comunicación serie entre el teclado y el ordenador, la comunicación de datos a través de un modem o la conexión de periféricos a través de una interfaz USB son implementaciones distintas del mismo concepto.

En este capítulo trataremos esencialmente de la comunicación serie implementada alrededor del Emisor Receptor Asíncrono Universal (UART) 8250/16450/16550. Este chip ha de realizar muchas funciones para posibilitar la comunicación serie, y tiene una estructura complicada, por lo que no descenderemos a los detalles de diseño, sino que nos centraremos en su estructura lógica e interfaz de programación. La comprensión de lo que sigue requiere antes presentar un par de conceptos.

5.2. Unidad simple de datos

La interfaz serie tiene gran importancia en la mayoría de los computadores debido a su gran flexibilidad, que permite la conexión de periféricos como

modems, plotter, ratones o impresoras. Al contrario que en la interfaz paralela, donde cada bit de un octeto de datos es transmitido por una línea física separada, en una interfaz serie los bits individuales de un octeto de datos, más un eventual bit de paridad, son transmitidos sucesivamente a través de una línea única.

Por otra parte, la transferencia serie puede ser síncrona o asíncrona. En el primer caso, una o varias señales adicionales son transmitidas, indicando cuando está disponible el siguiente bit en la línea de datos. Imaginemos una posible implementación. Una línea alterna entre los estados '0' y '1', y cada transición entre un valor y otro indica al receptor la existencia de un bit de datos en la línea de datos. El receptor consulta esta línea y obtiene el valor. Sin embargo, en una transferencia asíncrona se añade a los propios datos una mínima información de sincronización, de forma que el receptor puede distinguir entre bits individuales. Esta información adicional está formada normalmente por un *bit de inicio* y un *bit de parada*, frecuentemente precedido por un *bit de paridad* que permite una elemental detección de errores. Al conjunto de bit de incio, bits de datos, bit de paridad y bit de parada es a lo que llamamos *unidad simple de datos*, en adelante USD. Una USD contiene una sobrecarga de aproximadamente un 37 % respecto a los datos originales.

5.3. Paridad y baudios

Como acaba de decirse, el bit de paridad provee una elemental detección de errores [1]. Su ventaja es la simplicidad, y que todos los chips de interfaz serie pueden generarlo por hardware. El bit de paridad puede ajustarse de tal forma que en la USD aparezcan un número par o impar de unos, pero también puede fijarse a un valor dado. De esta forma puede detectarse un error en el propio bit de paridad, aunque sea a costa de no poder detectar la corrupción completa de los datos. Esto no parece muy inteligente.

Otro concepto necesario es el de velocidad en baudios. Es el número de veces por segundo que cambia el estado de la línea. Cuando ésta tiene sólo dos estados físicos, la velocidad en baudios coincide con la velocidad de transmisión en bits por segundo. En general, si n es el número de niveles físicos de la línea, v la velocidad en baudios y t la velocidad de transmisión en bits por segundo, se cumple que $t = v\log_2 n$.

[1] detecta errores simples, pero pasa por alto el 50 % de los errores de más de un bit

5.4. Serialización

Una USD puede admitir entre cinco y ocho bits de datos. Estos vienen precedidos por un bit de inicio, siempre a cero, y seguidos por un bit de paridad opcional y uno o varios bits de parada. La composición de una USD la lleva a cabo la UART, y para ello ha de valerse de varios registros. Al menos, uno para almacenar el octeto entrante y otro para componer, bit a bit, el paquete que será enviado. En cuanto a la recepción, será necesario al menos un registro para construir el paquete que se está recibiendo, y otro para depositar los datos extraidos del primero. Nótese que ambos extremos han de acordar el formato de la USD para que la decodificación tenga éxito. Por otra parte, ambos extremos han de funcionar a la misma velocidad, puesto que un bit se define por su *duración*. Ayuda a detectar esta duración el hecho de que el bit de inicio siempre valga '0', y que el bit de parada siempre valga '1', de tal forma que la llegada de una nueva USD se anuncia con una transición '1' a '0'.

Ahora bien, la serie de ceros y unos que forman un paquete suelen representarse como señales cuadradas, con bordes bien definidos. La realidad no es así. Si conectásemos un osciloscopio a una línea serie durante una transmisión, observaríamos bordes irregulares, y niveles lejos de ser constantes. Por este motivo cada bit es muestreado dieciseis veces, y del valor medio se deduce si corresponde a un '0' o a un '1'.

5.5. La interfaz de programación

5.5.1. Direcciones base

La UART 8250 tiene siete registros, y la 16450/16550 uno adicional. A estos diez registros se accede consecutivamente a partir de una dirección base. Existen hasta cuatro interfaces serie en el PC, y sus direcciones base se recogen en la tabla siguiente. Como puede verse, las parejas COM1-3 y COM2-4 comparten interrupción, lo que puede conducir a conflictos entre periféricos.

Interfaz	Dirección base	IRQ
COM1	0x3f8	IRQ4
COM2	0x2f8	IRQ3
COM3	0x3e8	IRQ4
COM4	0x2e8	IRQ3

5.5.2. Los buffers de recepción y transmisión

Con desplazamiento 0x00 a partir de la dirección base se puede acceder al *buffer* de recepción/transmisión. Después de que una USD ha sido convertida en un octeto útil por la UART, este octeto es colocado en este registro, donde queda a disposición del sistema. Téngase en cuenta que si se ha definido la longitud de un octeto con un número de bits inferior a ocho, los bits de orden superior en este registro están sin definir, aunque la instrucción `inportb` que usemos para leer ese octeto *siempre* tomará ocho bits. La siguiente línea de código lee un octeto recien llegado a través de COM1:

```
unsigned char octeto;
octeto=inportb(0x3f8);
```

Una vez realizada la lectura, el registro queda vacío, dispuesto para recibir nuevos datos. En cuanto a la transmisión, usa el mismo puerto. El octeto a transmitir es depositado allí y la UART se ocupa de insertar el bit de inicio, el bit de paridad y el bit de parada, formando una USD, y de transmitirla bit a bit. La complejidad a nivel hardware de la UART se traduce por tanto en una notable simplicidad de la interfaz de programación. El siguiente fragmento de código envía el carácter `'a'` a través de COM1:

```
outportb(0x3f8,'a');
```

5.5.3. El registro de habilitación de interrupciones

Hemos mencionado que la UART tiene dos bloques funcionales básicos. Uno encargado de serializar un octeto saliente, insertándolo en una USD, y otro encargado de reconstruir un octeto de datos entrante a partir de una USD. La UART contiene otros bloques funcionales, y uno de ellos es el encargado de la comunicación con un equipo de modulación/demodulación de señales, *modem*. La interfaz con este tipo de equipos está fijada mediante la norma RS-232, de la que nos ocuparemos más adelante. De momento baste indicar que la señal generada por la UART es inapropiada para ser enviada a través de líneas ruidosas y a grandes distancias, como suele suceder cuando se desea interconectar equipos. Por ese motivo, esta señal se usa para modular otra señal de características más apropiadas. Y de este trabajo se ocupa el modem. Es por tanto necesario que exista una comunicación entre la UART y el modem, de forma que pueda establecerse un protocolo entre ellos.

Volvamos al registro de habilitación de interrupciones. Cuando llega un octeto a la UART, debería producirse una interrupción, pues como sabemos la gestión por encuesta es poco eficiente. Pero también debería producirse una interrupción cuando se produjese alguna condición de error, de forma que la rutina que esté sirviendo la comunicación pueda comunicar esta circunstancia a las capas superiores. Para eso está el registro de habilitación de interrupciones.

Los cuatro bits altos de este registro siempre se encuentran a cero, y los cuatro bits inferiores instruyen a la UART para que genere una interrupción en determinadas circunstancias. Por ejemplo, el bit 0 a 1 indica la llegada de un octeto al *buffer* de recepción. El bit 1 a 1 indica a la UART que genere una interrupción cuando haya creado una USD a partir de un octeto saliente, y de esta forma la rutina que sirve los datos sabe que puede colocar el octeto siguiente.

El bit 2 a '1' instruye a la UART para que genere una interrupción en caso de error, y el bit 3 a '1' para que genere una interrupción cuando una de sus líneas de comunicación con el modem cambie de estado. De esta forma, una caida de portadora en la línea que comunica al modem con el mundo exterior sería detectada por éste, que a su vez lo comunicaría a la UART, que generaría una interrupción para que, finalmente, la rutina que sirve dicha interrupción pudiese comunicar esta circunstancia al programa de aplicación que está enviando los datos.

El registro de habilitación de interrupciones se encuentra en el desplazamiento 0x01 a partir de la dirección base. El siguiente fragmento de código configura la UART de forma que se produzca una interrupción cuando un octeto entrante está disponible para leer, o cuando un octeto saliente ha dejado libre el *buffer* de transmisión en la interfaz COM1:

```
unsigned char octeto;
octeto=inportb(0x3f9);   /* direccion base mas uno */
octeto |= 3;             /* activamos bits 0 y 1 */
outportb(0x3f9,octeto);  /* configuramos UART */
```

5.5.4. El registro de identificación de interrupción

Hemos dicho que la gestión por encuesta es poco eficiente. Sin embargo, a veces es necesaria, y la UART es suficientemente flexible como para apoyar

este tipo de gestión. Para ello, provee de un registro especial. Cuando se produce un evento que normalmente generaría una interrupción, tal como se configuró mediante el registro explicado en la sección anterior, un bit del registro de identificación de interrupción se pone a cero, y dos bits adicionales indican cual es la causa que requiere atención. De esta forma, un programa de aplicación sólo tendría que leer constantemente en un bucle este registro. El bit 0 a '0' indicaría una circunstancia que requiere atención. Entonces, el programa consultaría los bits 1 y 2, y averiguaría de que se trata. La codificación de los bits 1 y 2 es la siguiente: '00' (3) cambio en una línea RS-232; '01' (2) *buffer* de transmisión vacio; '10' (1) octeto recibido; '11' (0) error.

Es importante notar aquí que la UART tiene predefinidas prioridades para cada una de las circunstancias anteriores. Estas prioridades se han escrito entre paréntesis. Cuando una de ellas está siendo atendida, las de prioridad inferior son bloqueadas, de manera que la rutina que se encargue de su tratamiento ha de tener alguna forma de devolver a la UART a un estado en que pueda generar una interrupción de cualquier prioridad. De igual forma que las prioridades de las interrupciones están predefinidas, también lo están las acciones que desbloquean las interrupciones inferiores a una dada. Estas acciones son las siguientes:

1. Cambio en una línea RS-232: Leer el registro de estado del RS-232
2. Registro de transmisión vacío: Escribir nuevo octeto para transmitir o leer el registro de identificación de interrupción
3. Byte recibido: Leer octeto
4. Error: Leer el registro de estado de serialización

El siguiente sencillo fragmento de código explora contínuamente el registro de identificación de interrupción para COM1, que se encuentra en el desplazamiento 0x02 a partir de la dirección base:

```
unsigned char octeto;
while (1){
    octeto=inportb(0x3fa); /* desplazamiento 2 a partir base */
    if (octeto ^ 1){
        /* identificar causa y actuar en consecuencia */
    }
}
```

5.5.5. Registro de formato de datos

Este es el registro que permite especificar el formato para la USD. Se encuentra en el desplazamiento 0x03 a partir de la dirección base. Así, los bits 0 y 1 indican el número de bits de datos dentro de la USD: '00' para 5 bits de datos, '01' para seis, '10' para siete y '11' para ocho. El bit 2 indica si se introducirán uno ('0') o dos ('1') bits de parada. Los bits 3-4-5 indican la paridad: '000' sin paridad, '001' par, '011' impar, '101' fija a 1, '111' fija a cero.

El bit siete requiere una explicación especial. Cuando toma el valor de '1' indica que puede accederse a los registros situados en los desplazamiento 0x00 y 0x01 para escribir los octetos bajo y alto respectivamente del divisor de frecuencia que permite establecer la velocidad de operación en baudios. Esta velocidad viene dada por

$$v = \frac{115200}{divisor} \tag{5.1}$$

De esta manera, un divisor a 1 permite transmisión a 115200 baudios. Con un bit de inicio, ocho bits de datos, sin paridad, y un bit de parada, esto significa 11520 octetos por segundo, lo que deja a la CPU 86 μs para leer un octeto. El siguiente fragmento de código configura COM1 para que opere a 300 baudios. El divisor ha de tomar entonces el valor 384:

```
unsigned char octeto;
octeto=inportb(0x3fb);   /* leer registro de formato */
octeto |= (1<<7);        /* activar bit siete */
outportb(0x3fb,octeto);  /* escribir */
outportb(0x3f8,0x80);    /* octeto bajo del divisor */
outportb(0x3f9,0x01);    /* octeto alto del divisor */
octeto=inportb(0x3fb);   /* leer registro de formato */
octeto ^ = (1<<7);       /* desactivar bit siete */
outportb(0x3fb,octeto);  /* escribir */
```

5.5.6. Registro de control de modem

Este registro controla la interfaz con el modem. Se encuentra en el desplazamiento 0x04 y contiene algunos bits interesantes. El cuatro, por ejemplo, permite trabajar con el 8250 en bucle cerrado, lo que es muy interesante durante el desarrollo de software de comunicaciones. En este modo, un programa escribe un octeto en el registro de transmisión, y una vez procesado

por la UART se genera una interrupción. Este octeto es devuelto al registro de recepción, donde el programa, alertado por la consiguiente interrupción, puede leerlo y comprobar la correspondencia entre lo que se envía y lo que se recibe.

Por otro lado, un valor de '1' en el bit número 3 inhibe las interrupciones de la interfaz serie. Finalmente, son de interés los bits 0 y 1, ya que son usados para el *diálogo* entre la UART y el modem.

5.5.7. Registro de estado de serialización

Este registro contiene información sobre el estado de las secciones de recepción y transmisión de la UART. El bit más significativo no se usa, y siempre está a cero. El bit número 6 a uno indica que los dos registros de transmisión (el que guarda el octeto que se desea enviar y el usado para serializar los datos) están vacíos. A cero, indica que alguno de los dos se encuentra ocupado. Un valor '1' en el bit número 5 indica que el registro de transmisión está ocupado.

Los bits 4, 3, 2 y 1 se usan para indicar distintas condiciones de error, y el bit 0 a 1 indica la existencia de un octeto disponible en el *buffer* de recepción.

El registro de estado de serialización, junto con el registro de identificación de interrupción, permite a un programa que funcione por encuesta determinar exactamente la causa de un error, o determinar el estado de la UART incluso si las interrupciones están deshabilitadas.

5.5.8. Registro de estado del modem

Este registro permite determinar el valor de cuatro variables de estado del modem, y si estos valores han cambiado desde la última vez que fueron consultados. Diferimos la explicación de estos valores para la futura discusión de la interfaz RS-232.

5.5.9. Registro auxiliar

Existe en las implementaciones 16450/16550, pero no en la original 8250. Se conoce como *scratch-pad* en la literatura, y puede usarse como una celdilla de memoria disponible para el programador. Su valor no afecta en ningún modo al funcionamiento de la UART.

5.6. Un programa de ejemplo

El siguiente es un programa de ejemplo tomado directamente de .[El] Universo Digital del IBM PC, AT y PS/2", pag. 351, con algunas simplificaciones de aspectos no relevantes para nuestra discusión. Ilustra el autodiagnóstico del 8250 en modo lazo cerrado:

```
#include <dos.h>
#include <conio.h>

#define LCR (base+3) /* registro de control de linea */
#define IER (base+1) /* registro de activacion de interrupciones */
#define DLL (base+0) /* parte baja del divisor */
#define DLM (base+1) /* parte alta del divisor */
#define MCR (base+4) /* registro de control del modem */
#define LSR (base+5) /* registro de estado de linea */
#define RBR (base+0) /* registro de recepcion */
#define THR (base+0) /* registro de retencion de transmision */

#define DR   1    /* dato disponible del LSR */
#define OE   2    /* bit de error de overrun del LSR */
#define PE   4    /* bit de error de paridad del LSR */
#define FE   8    /* bit de error de bits de stop del LSR */
#define BI   0x10 /* bit de error de break del LSR */
#define THRE 0x20 /* bit de THR vacio */

void error()
{
    printf("error del puerto serie\n");
    return(2);
}


void main()
{
    unsigned com, base, divisor, dato, entrada, lsr;

    /*
    aqui, un fragmento de codigo se ocuparia de averiguar
    com, base y divisor, preguntando al usuario.
    */
```

```
    outportb(LCR,0x83)          /* DLAB=1, 8 bits, 1 stop,
                                   sin paridad */
    outportb(IER,0);
    outportb(DLL,divisor % 256);
    outportb(DLM,divisor>>8);
    outportb(MCR,8+16);         /* modo loop */
    outportb(LCR,0x03);         /* DLAB=0, 8 bits, 1 stop,
                                   sin paridad */

    for(dato=0;dato<0x100 && !kbhit();++dato){

        do{ /* espera THR vacio */
            lsr=inportb(LSR);
            if (lsr & (OE|PE|FE|BI)) error();
        }while (!(lsr & THRE));
        outportb(THR,dato);

        do{ /* espera RBR lleno */
            lsr=inportb(LSR);
            if (lsr & (OE|PE|FE|BI)) error();
        } while (!(lsr & DR))

        entrada=inportb(RBR);
        if (dato!=entrada) error();

        if (!kbhit()){
            printf("diagnostico superado");
            return(0);
        }
    }
}
```

5.7. La interfaz V.24

5.7.1. Características generales

La mayoría de los ordenadores personales usan la interfaz V.24 para la comunicación vía serie. Esta interfaz [2], define las características mecánicas,

[2]Conocida en Estados Unidos como RS-232

eléctricas y lógicas de la conexión entre un Equipo Terminal de Datos (ETD) y un Equipo Portador de Datos (EPD). Normalmente, el primero es un PC, y el segundo un modem.

V.24 define una conexión de 25 líneas entre el primero y el segundo, y por ende una implementación física con 25 patillas, donde la mayoría están reservadas para transferencia serie síncrona. Para transferencias serie asíncronas sólo 11 son necesarias. IBM por su parte define su propia interfaz de sólo 9 líneas, muy extendida. La siguiente tabla muestras las distintas conexiones en las implementaciones de 25 y 9 pines, su denominación y descripción:

25 pines	**9 pines**	**Señal**	**Dirección**	**Descripción**
1	-	-	-	Tierra
2	3	TD	PC-MO	Transmisión
3	2	RD	MO-PC	Recepción
4	7	RTS	PC-MO	Petición de envío
5	8	CTS	MO-PC	Preparado para aceptar datos
6	6	DSR	MO-PC	Preparado para comunicarse
7	5	-	-	Tierra
8	1	DCD	MO-PC	Portadora detectada
20	4	DTR	PC-MO	Interruptor general
22	9	RI	MO-PC	Indicador de llamada
23	-	DSRS	MO*PC	Selección de velocidad

Para fijar ideas, hemos abandonado la nomenclatura oficial para adoptar una más inmediata. Así, al equipo terminal de datos le llamamos simplemente 'PC', y al equipo portador de datos, simplemente 'MO' (modem). Las tres líneas obvias son la Tierra común, la línea para enviar datos al modem y la línea para recibir datos del modem. El resto de líneas sirven para arbitrar un protocolo de comunicación entre uno y otro. Por ejemplo, 'RI' (ring) permite al modem comunicar al PC una llamada entrante, y DSRS permite al modem comunicar al PC la velocidad a la que está recibiendo datos del exterior, que puede ser distinta a la velocidad de comunicación entre la UART y el modem. La cuarta columna indica la pareja origen-destino de la comunicación, y el asterisco en la última línea indica comunicación bidireccional.

¿ Cómo se implementa el protocolo de comunicación entre el PC, a través de su UART, y el modem ?. Para empezar, existe una especie de *interruptor general*, implementado mediante la línea DTR. Una vez activada esta línea, el PC usa RTS para indicar al modem que desea comunicar datos con el

mundo exterior. Como respuesta, el modem activará su frecuencia portadora hacia el exterior (p. ej., introducirá esta frecuencia a través del conector de la línea telefónica). Cuando el modem está listo para recibir datos del PC, lo comunica a través de la línea CTS. La comunicación podrá existir en tanto en cuanto exista portadora, y en las líneas telefónicas convencionales no existe garantía real de continuidad de esta señal, por lo que el modem implementa la línea DCD mediante el cual comunica al PC la existencia de esta señal.

Por su parte, DSR es similar a CTS, salvo que el modem indica además que ha terminado los preparativos para la comunicación con el equipo situado al otro lado de la línea.

5.7.2. Tipos de conexión

Con el apoyo hardware expuesto en el punto anterior, pueden implementarse varios tipos de comunicación entre el PC y el modem. El más sencillo es el conocido como comunicación *simplex*. El PC envía datos al modem a través de la línea TD. RD está inactiva, y tanto RTS como CTS se mantienen inactivas o siempre en el mismo estado. De la misma forma, DCD se encuentra inactiva, puesto que el modem sólo puede recibir datos del PC, y sólo comunicarlos hacia el mundo exterior.

En una comunicación *semi-duplex* puede darse comunicación bidireccional, pero no simultánea, entre el PC y el modem. Finalmente, en una comunicación *duplex-completa* la comunicación es bidireccional y simultánea.

Esta es una posible secuencia de sucesos cuando una llamada procedente del mundo exterior llega al modem:

- Justo antes de la llamada, RI y DCD se encuentran inactivos.
- La llegada de la llamada externa al modem es detectada por éste, que activa la línea RI para indicar al PC el evento.
- El PC detecta RI y activa DTR para indicar su disposición, activar el modem e indicarle que se prepare para aceptar la llamada. Por su parte, el modem ha de establecer la conexión con el emisor.
- Una vez el modem ha terminado sus preparativos, activa DSR.
- Si la conexión entre el modem y el mundo exterior se ha establecido, éste activa la línea DCD para informar al PC de que la conexión es usable, y de que el PC puede enviar y recibir datos.

- Mediante RTS y CTS se regula la comunicación entre PC y modem.
- La comunicación se mantiene mientras se encuentren activos DCD, DSR o DTR.

Como puede verse, existe un apoyo hardware suficiente para implementar una amplia variedad de protocolos serie. Existen muchos ejemplos en la literatura, que nosotros no vamos a explorar. Por citar algunos, las conexiones de impresoras vía serie o la conexión entre dos ordenadores para intercambio de datos usando un cable *null-modem*. Y esto, sin abandonar la interfaz V.24, ya que existen otras implementaciones de comunicaciones serie.

5.7.3. De vuelta a los registros de la UART

En la exposición sobre los registros de la UART diferimos la explicación del Registro de Estado del Modem. Este registro indica en sus 4 bits altos los estados de DCD, RI, DSR y CTS; en los cuatro bits de orden inferior, un 1 indica si ha existido alguna variación en el bit correspondiente de orden superior desde la última consulta.

Unas últimas palabras sobre los modems. Estos periféricos realizan operaciones de mucho más alto nivel de las que hemos expuesto aquí, como, por ejemplo, llamar a un determinado número de teléfono, colgar si se supera un número determinado de *ring*, repetir la última llamada, etc. Hasta ahora existía un *standard de facto* creado por la compañia Hayes para sus modems, pero la quiebra reciente de la misma provoca una situación incierta.

5.8. Dispositivos apuntadores

5.8.1. Ratones

El representante más común de los dispositivos apuntadores es el conocido ratón, aunque existen otros dispositivos similares, como las tabletas digitalizadoras o las bolas de seguimiento (*trackballs*). Este tipo de dispositivos comunican datos vía serie, según una variedad de protocolos. Nos centraremos en los dos tipos mas comunes de ratón: Serie y PS/2.

Físicamente, un ratón es un dispositivo que mantiene una esfera rugosa en contacto con una superficie. Esta esfera puede hacerse rodar, y su giro es detectado por unos sensores, codificado, y procesado por un microchip antes

de ser enviado, en un formato especial, al ordenador. Este microchip realiza tareas relativamente complejas, pues no sólo detecta el movimiento, sino que calcula la velocidad de dicho movimiento y de esta forma puede distinguir entre los dos tipos de desplazamientos que se realizan con el puntero. En efecto, observando a personas que hacen uso del ratón e interactúan con programas que lo toman como entrada natural puede verse que se realizan, por una parte, desplazamientos rápidos de largo recorrido, y por otra desplazamientos lentos de aproximación. Por ejemplo, se desplaza rápidamente el ratón para alcanzar un área de la pantalla, y después se hace un movimiento más fino para apuntar a un botón. Estas diferencias de velocidad son detectadas por el micro, que genera incrementos no lineales en las coordenadas, para adaptarse al uso que se esté dando al ratón. Para fijar ideas, un movimiento lento de dos centímetros sobre una mesa se refleja en un desplazamiento de seis centímetros sobre la pantalla, pero si la velocidad de desplazamiento es grande, el micro supone que el usuario desea hacer un desplazamiento largo, y *engaña* al sistema transmitiéndole mayores incrementos en las coordenadas de las que realmente proporcionaron los sensores, como si la bola hubiese girado más de lo que realmente lo ha hecho. El resultado es que el sistema asigna al puntero sobre la pantalla unas coordenadas más próximas al punto al que el usuario se dirige.

Por lo demás, un ratón puede acomodar entre uno y tres botones, y también es preciso indicar el estado de cada uno. De forma que cuando se produce un movimiento en el ratón, o una pulsación, el ratón ha de generar un paquete de datos, que serán enviados vía serie y procesados normalmente por una rutina de servicio de interrupción, ya que no tiene mucho sentido hacer un tratamiento por encuesta de un dispositivo como éste.

La mayoría de los ratones se conecta a 1200 baudios, con 7 bits de datos y sin paridad. La detección del ratón se realiza colocando a 1 la línea DTR del puerto serie. Tras unos instantes, el ratón devolverá el carácter ‘M’. A partir de aquí, los movimientos o eventos de botones provocan ráfagas de tres interrupciones, mediante las cuales el ratón indica al controlador los desplazamientos en los ejes ‘X’ e ‘Y’, así como el estado de los botones. Estos desplazamientos se encuentran en el rango -127 a 128. La figura siguiente muestra el formato de estos tres octetos.

Obsérvese que el bit alto del primer octeto se encuentra a 1, mientras que los bits altos de los octetos segundo y tercero se encuentra a cero. El propósito de esto es el de permitir al controlador software recuperar la sincronización,

caso de producirse algún tipo de error. Por su parte, los bits 'L' y ' R' indican el estado de los botones izquierdo y derecho.

Este sencillo protocolo está siendo sustituido rápidamente por el protocolo PS/2, desarrollado por IBM para sus ratones. El protocolo PS/2 se apoya en el controlador de teclado, usando sus mismos puertos, y el ratón envía paquetes de ocho octetos, pudiendo también aceptar hasta dieciseis órdenes distintas que permiten conmutar entre gestión por interrupciones y encuesta, redefinir el factor de escala entre los ejes 'X' e 'Y' o inducir una comprobación interna del ratón. Otra interesante capacidad consiste en cambiar el modo de funcionamiento a *remoto*. En este modo, el ratón no envía datos de forma autónoma, sino que espera un comando específico para enviarlos. Por supuesto, la velocidad de muestreo cuando funciona en forma autónoma también puede alterarse enviando la orden correspondiente. [3]

Tanto el protocolo serie como el protocolo PS/2 suponen ratones de dos botones, pero sabemos que existen también con tres botones. En estos casos, el fabricante ofrece el controlador software adecuado para procesar el formato propio del paquete enviado por su ratón. En el otro extremo, la gestión del ratón puede hacerse también con un solo botón, como demuestra Apple en sus populares MacIntosh.

[3]La información detallada puede encontrarse en las páginas 1036 a 1039 de "The indispensable PC hardware book", Addison-Wesley. Tercera edición de 1998, o cuarta edición, más reciente.

5.8.2. Otros dispositivos apuntadores

Una bola de seguimiento, llamada *trackball* por aquellos que prefieran el término anglosajón, tiene la misma estructura que un ratón normal, salvo que la esfera que rueda no lo hace indirectamente, arrastrada por el movimiento del ratón, sino que es accionada directamente por los dedos del usuario. Esencialmente, se trata de un ratón *boca arriba*, con una esfera de mayor tamaño, para facilitar su uso. Su interfaz hardware es un puerto serie y su interfaz software es la misma que la de un ratón convencional.

Cualquiera que haya tratado de usar un ratón para trabajos que requieran cierta precisión, como retoque fotográfico, habrá experimentado la falta de precisión de este dispositivo. [4] Para aquellas aplicaciones que requieren movimientos precisos, y no sólo posicionamiento preciso, existen las tabletas digitalizadoras, especialmente adaptadas al mundo del CAD. Una tableta digitalizadora consta esencialmente de una superficie plana donde es posible mover un sensor. Bajo esta superficie existe una densa cuadrícula de conductores. El hardware de la tableta genera pulsos sucesivamente para cada conductor en el eje 'X', y a su vez sucesivamente para cada conductor en el eje 'Y' que cruza el primero. Estos pulsos por tanto van recorriendo toda la cuadrícula. Cuando el sensor se encuentra sobre uno de los cruces, puede, a partir del tiempo en que se detecta el pulso, determinar precisamente su posición, y enviarla al software correspondiente.

[4] especialmente evidente cuando se quiere escribir algo.

Capítulo 6

Impresoras

6.1. Introducción

Los dispositivos de impresión ocupan un papel destacado en el conjunto de los periféricos asociados a los sistemas de computación. Aparte de satisfacer necesidades prácticas, permiten obtener copias permanentes de la información que se desee; copias generalmente más seguras que aquellas que se hacen en soporte magnético.

Normalmente, las impresoras se conectan al sistema de computación mediante el puerto paralelo, pero también mediante puerto serie o interfaz de red. El primer caso es adecuado para impresoras físicamente asociadas a un computador. El segundo, cuando la distancia es mayor que unos pocos metros. La interfaz de red es útil cuando una impresora ha de dar servicio a una pequeña red de área local, como la de un departamento o una empresa.

6.2. Tecnologías de impresión

Existe una gran variedad de tecnologías de impresión. Cada una de ellas es adecuada para unas necesidades concretas, por lo que no puede decirse que sean excluyentes entre sí, sino que en cada caso será preciso valorar el uso que se hará del sistema de impresión. Por ejemplo, la tecnología de impacto es imprescindible cuando se ha de satisfacer una elevada carga de trabajo, gran velocidad y bajo coste, aunque a expensas de unas posibilidades más limitadas. Para uso personal, con carga baja y uso ocasional del color, una impresora de inyección de tinta parece la elección más adecuada, mientras que para un uso profesional, con carga medio-baja lo más interesante puede ser una impresora *laser* de gama media con soporte PostScript.

6.2.1. Tecnología de impacto

Es la empleada por las impresoras matriciales. Este sistema se basa en un cabezal de impresión cuya parte principal es una fila de agujas que pueden accionarse mediante pequeños electroimanes, haciendo que golpeen contra una cinta impregnada en tinta, que a su vez se encuentra a una pequeña distancia del papel. Tienen la ventaja del bajo coste de impresión por página, bajo coste de fungibles y posibilidad de usar papel autocopiativo para imprimir varias copias simultáneamente. La velocidad de impresión puede ser elevada, de hasta miles de caracteres por minuto, y puede aumentarse disponiendo, en lugar de un cabezal que se desplaza de izquierda a derecha a medida que compone la línea, de una fila de cabezales, tantos como caracteres de ancho tenga el documento que va a imprimirse, que golpean simultáneamente, componiendo una línea entera cada vez. A estas impresoras se les llama *impresoras de líneas*, y suelen encontrarse en grandes centros de cálculo, que precisan lanzar un volumen elevado de documentos de tipo específico: listados de programas, listados administrativos y similares.

6.2.2. Tecnología de inyección de tinta

Las impresoras de inyección de tinta tienen un importante hueco en las aplicaciones personales. Suelen ser impresoras de bajo coste cuando se usan en este ambiente, pero al mismo tiempo pueden ser impresoras caras y complejas cuando se usan en ambientes profesionales o semiprofesionales donde se imprime en color.

El cabezal de una impresora de inyección contiene una o varias filas de orificios por donde la tinta es expelida. Esta procede de un depósito que puede estar unido o no a los inyectores. En cuanto al mecanismo de inyección, suele basarse en una pequeña cámara conectada directamente a los orificios y cuyo volumen puede variarse mediante impulsos eléctricos, bien provocando una pequeña deformación neumática o bien aprovechando un efecto piezoeléctrico. A su vez, cada uno de los orificios del cabezal, de un diámetro que es una pequeña fracción de milímetro, puede controlarse individualmente en su apertura mediante impulsos eléctricos. En otros diseños, se dota a las pequeñas gotitas de una carga eléctrica superficial, lo que se aprovecha para inducir o inhibir su salida a través de los inyectores.

6.2.3. Tecnología laser

La tecnología *laser* se basa en la propiedad de ciertas sustancias para mantener una carga eléctrica superficial localizada. Cuando se dispone de un rodillo de este material, puede, mediante la iluminación con luz *laser*, inducirse o no en cada punto de su superficie una carga eléctrica. De esta forma, en los puntos que se desee puede adherirse una sustancia fínamente pulverizada, que después pasa por contacto a la hoja. Allí se fija mediante fusión por calor.

La tecnología *laser* es la que probablemente abarca un mayor rango de aplicaciones, y así encontramos desde impresoras domésticas diseñadas para cargas bajas de trabajo hasta grandes sistemas de impresión, diseñados para efectuar decenas de miles de copias mensuales a velocidades de decenas de páginas por minuto, tanto con papel continuo como con hojas individuales.

6.2.4. Tecnología de sublimación de tinta sólida

En determinadas aplicaciones de diseño donde se requiere color, la sublimación de tinta sólida es una buena opción, aún cuando este tipo de impresoras se caracterizan por su elevado precio, asi como por el precio también alto de los fungibles. En este tipo de impresoras, una sustancia, la tinta en estado sólido, bajo determinadas condiciones físicas pasa directamente a estado gaseoso, fijándose electrostáticamente al soporte, para después solidificarse.

6.3. Programación del puerto paralelo

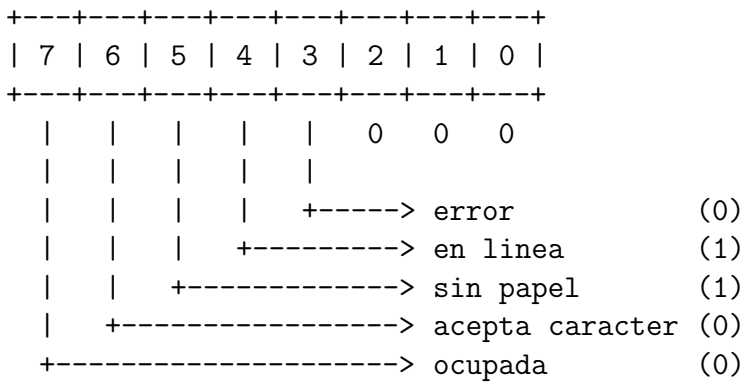
6.3.1. Estructura de los registros

Los puertos paralelos en la arquitectura PC tienen asignados rangos fijos de direcciones E/S. Cada puerto tiene asignadas tres direcciones consecutivas, y estas direcciones pueden consultarse en la zona de datos de la BIOS, concretamente en:

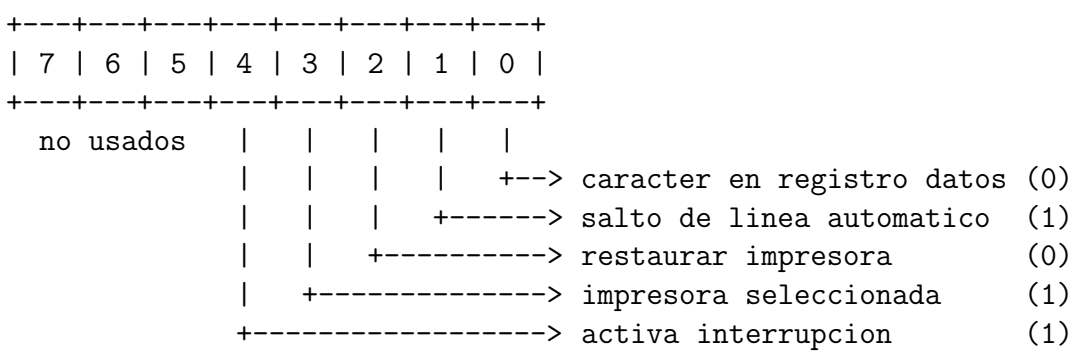
Dirección base	Interfaz
0x40:0x08	primera
0x40:0x0A	segunda
0x40:0x0C	tercera
0x40:0x0E	cuarta

Estas tres direcciones forman una interfaz unificada para controlar el puerto a nivel software. El primer octeto se corresponde con los datos que van a transmitirse, y cada uno de sus bits se transmite a través de uno de los hilos llamados D0-D7 en el cable paralelo. Este registro es de sólo escritura.

El segundo octeto es de sólo lectura, y es el octeto de estado de la impresora. El significado de cada uno de sus bits es el siguiente:



El tercer registro sirve para el control de la impresora, y contiene un bit con cuya ayuda puede provocarse una interrupción hardware. Este registro es de sólo escritura, y su esquema es el siguiente:



6.3.2. Estructura de la comunicación

Usaremos la notacion x/y para indicar el bit 'y' del registro 'x' de un puerto en particular. La comunicación comienza poniendo un octeto de datos en el primer registro. A continuación, es preciso indicar a la impresora que puede procesar este dato, lo que se consigue poniendo a cero el bit 2/0. Una particularidad de este bit es que debe ser activado de nuevo inmediatamente, pues de lo contrario la impresora podría leer varias veces el mismo carácter.

Pero, ¿ cuanto tiempo es preciso dejarle a la impresora para que lea el carácter ? He aquí un diseño poco afortunado. Inmediatamente después de recibir la señal asociada al bit 2/0, y dada la lentitud de la impresora en procesar un carácter, ésta desactiva el bit 1/7, con lo que bloquea nuevas transmisiones de datos. Simultáneamente, la impresora ha de poner a 1 el bit 1/6.

Considérese la siguiente rutina para enviar un carácter a la impresora. Esta rutina devuelve un código de error:

```
int salida_imp(char caracter)
{
    int i,codigo;
    for(i=0;i<TIMEOUT;++i){      /* leer el registro de     */
        codigo=inp(0x379);       /* estado, y esperar hasta */
        if((codigo & 0x80)==0x80)/* que la impresora no este*/
            break;               /* ocupada o venza contador*/
    }
    if (i==TIMEOUT) return(1);

    outp(0x378,caracter);        /* escribir caracter       */

    codigo=0x37a;                /* poner a 1 el bit        */
    codigo |= 0x01;              /* STROBE para indicar el  */
    outp(0x37a,codigo);          /* envio del caracter      */

    for(i=0;i<STROBEWAIT;++i);   /* dar cierto tiempo a la  */
                                 /* impresora para que lea  */
    codigo=inp(0x37a);           /* desactivar el bit       */
    codigo &= 0xfe;              /* STROBE para que no se   */
    outp(0x37a,codigo);          /* produzcan mas lecturas  */

    for(i=0;i<TIMEOUT;++i){      /* esperar un tiempo a la  */
        codigo=inp(0x379);       /* impresora y comprobar   */
        if((codigo & 0x40)==0x00)/* el bit ACK              */
            break;
    }

    if (i==TIMEOUT) return(2);   /* error si la impresora   */
                                 /* no indico ACK           */

    return(0);                   /* codigo habitual cuando  */
```

```
                                    /* la operacion es buena   */
}
```

El fragmento de código siguiente ilustra en envío de caracteres a la impresora usando interrupciones. El procedimiento consiste en enviar el primer carácter mediante programa, configurando el puerto de manera que la disposición de la impresora para recibir nuevos caracteres se indique mediante una interrupción. A partir de este momento, la rutina que sirve dicha interrupción es la encargada de enviar caracteres sucesivos.

```
/*
Precauciones:
La BIOS utiliza el metodo programado para escribir en la
impresora, de manera que normalmente la capacidad del puerto
paralelo para generar una interrupcion esta desactivada.
Debemos entrar en la BIOS y habilitar el PP para que genere
interrupciones. Pero, consecuente con el estado por defecto,
la IRQ7 esta enmascarada, de manera que aunque la interrupcion
se genere, el PIC no la atendera. Por eso es preciso habilitar
la linea 7 del PIC, de lo que se encargan lineas 105-107.
*/

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

/*--- datos globales --- */
char buffer[]="Esto es una prueba para enviar\n a la impresora";
int contador=0;
int TIMEOUT=30000;
int TIMESTR=1000;

/*---------------------- enviar caracter a la impresora ----------------*/
int enviar(char c)
{
    int i,codigo;

    /* esperar hasta que la impresora este libre */
    for(i=0;i<TIMEOUT;++i){
        codigo=inportb(0x379);
        if ((codigo&128)==128) break;
```

```
    }
    if (i==TIMEOUT) return(1);

    /* poner el dato en el registro de datos */
    outportb(0x378,c);

    /* poner a 1 el bit STROBE */
    codigo=inportb(0x37a);
    codigo|=1;
    outportb(0x37a,codigo);

    /* darle un poco de tiempo a la impresora para que lea el caracter */
    for(i=0;i<TIMESTR;++i);

    /* poner a cero el bit STROBE */
    codigo=inportb(0x37a);
    codigo&=0xfe;
    outportb(0x37a,codigo);

    /* esperar a que la impresora de el ACK */
    for(i=0;i<TIMEOUT;++i){
        codigo=inportb(0x379);
        if ((codigo & 64)==0) return(0);
    }
    if (i==TIMEOUT) return(1);
}

/*--------- Servicio de interrupcion del puerto paralelo LPT1 ----------*/
void interrupt SII()
{
    int codigo,i;

    /* poner el dato en el registro de datos */
    if (buffer[contador]!='\0'){

        outportb(0x378,buffer[contador++]);

        /* poner a 1 el bit STROBE */
        codigo=inportb(0x37a);
        codigo|=1;
        outportb(0x37a,codigo);
```

```
        /* darle un poco de tiempo a la impresora para que lea el caracter */
        for(i=0;i<TIMESTR;++i);

        /* poner a cero el bit STROBE */
        codigo=inportb(0x37a);
        codigo&=0xfe;
        /* poner a 1 el bit IRQ */
        codigo|=16;
        outportb(0x37a,codigo);

    }

    /* restaurar PIC */
    outportb(0x20,0x20);
}

/*----- main --------*/
main()
{
    int *p,seg,des,j;
    char codigo;

    /* redireccionar servicio interrupcion LPT1 */
    p=(int *)MK_FP(0,60);
    des=*p;
    seg=*(p+1);
    asm cli;
    *p=FP_OFF(SII);
    *(p+1)=FP_SEG(SII);
    asm sti;

    /* iniciar el PIC para que acepte IRQ7 poniendo bit 7 a cero */
    codigo=inportb(0x21);
    codigo&=127;
    outportb(0x21,codigo);

    /* iniciar la impresora */
    codigo=inportb(0x37a);
    /* poner a 1 bits 2 y 4 */
    codigo|=20;
```

```
    outportb(0x37a,codigo);

    /* enviar el primer caracter */
    if (enviar(buffer[contador++])==1){
        printf("Error de impresora\n");
        return(0);
    }

    /* mientras se envia el resto de caracteres, hacer otra cosa */
    printf("imprimiendo en segundo plano\n");
    while (!kbhit()){
    }

    /* recuperar el antiguo vector de interrupcion */
    asm cli;
    *p=des;
    *(p+1)=seg;
    asm sti;

    printf("\n%d\n",contador);

}
```

6.3.3. Conexionado

Se presenta a continuación el conexionado de los diversos hilos que forman el cable paralelo. Existen en la bibliografía descripciones de como pueden alterarse las conexiones para conseguir propósitos específicos, distintos de la mera transmisión de caracteres. Por ejemplo, pueden construirse cables de los llamados *null modem* que permiten conectar dos ordenadores.

hilo ordenador	hilo impresora	nombre	significado
1	1	STROBE	Indica transmisión
2	2	D0	bit 0 de datos
3	3	D1	bit 1
4	4	D2	bit 2
5	5	D3	bit 3
6	6	D4	bit 4
7	7	D5	bit 5
8	8	D6	bit 6
9	9	D7	bit 7
10	10	-ACK	último caracter correcto
11	11	-BUSY	impresora ocupada
12	12	PE	impresora sin papel
13	13	SLCT	en línea
14	14	-AUTOFEED	RC automático trás SL
15	32	-ERROR	error
16	31	-INIT	restaurar impresora
17	36	SLCT IN	activar en línea
18-25	19-30	GND	tierra

Ahora bien, es importante darse cuenta de que la interfaz paralela está diseñada con la suficiente flexibilidad como para no estar necesariamente ligada a algún periférico en particular, en nuestro caso a una impresora. Así, el registro base está conectado a los hilos 2 al 9, y es un registro bidireccional que puede usarse para lectura/escritura en un contexto más general.

El registro situado en el desplazamiento 1 es un registro de sólo lectura, con el bit 7 conectado al hilo 11, invertido, el bit 6 al hilo 10, el bit 5 al hilo 12, el bit 4 al hilo 13 y el bit 3 al hilo 15. Los tres bits de orden inferior no se usan, y están normalmente a 1.

El siguiente registro, situado en el desplazamiento 2 a partir de la dirección base, es parcialmente bidireccional. El bit más interesante es el número 4. Cuando se pone a 1 se está indicando a la interfaz que genere una interrupción cuando el hilo 10 caiga del nivel alto al nivel cero. Los tres bits de mayor orden no se usan, y los bits 3, 2, 1 y 0 están unidos respectivamente a los hilos 17, 16, 14 y 1. Los hilos 17, 14 y 1 están invertidos; es decir, un valor a 1 en un bit invertido indica nivel bajo en el hilo correspondiente.

Con esta base, es posible usar la interfaz para menesteres distintos de los habituales. Por ejemplo, es fácil usarla para comunicar dos ordenadores de forma más eficiente que mediante la interfaz serie [1].

6.4. Funciones BIOS para control del puerto paralelo

La interrupción 0x17 de la BIOS está dedicada a la comunicación con el puerto paralelo. Al PC pueden conectarse un máximo de tres puertos paralelo, que pueden usarse mediante tres funciones distintas. No crea el lector, debido a la coincidencia entre número de puertos y número de funciones, que hay asignada una función para cada puerto. Las tres funciones que pone el BIOS a nuestra disposición son:

Función	Uso
0x00	Enviar carácter
0x01	Iniciar impresora
0x02	Preguntar estado impresora

En la llamada a cada una de estas funciones, se espera el número de puerto en DX, pudiendo pasar los valores de 0 a 2, que corresponden a LPT1, LPT2 y LPT3 en D.O.S..

Aparte del número de puerto en DX, las tres funciones tienen en común que devuelven en AH el estado de la impresora. El error de tiempo ocurre siempre que el BIOS intenta transmitir datos a la impresora durante un cierto tiempo pero está ocupada o no responde. La cantidad de intentos que realiza el BIOS antes de dar un error de tiempo depende del contenido de tres octetos que se encuentran a partir de la posicion 0x0040:0x0078:

Dirección	interfaz
0x0040:0x0078	contador del primer puerto
0x0040:0x0079	contador del segundo puerto
0x0040:0x0080	contador del tercer puerto

[1] Véase por ejemplo "The indispensable PC Hardware Handbook", página 940

Los valores que el BIOS pone en estas direcciones no indican un lapso de tiempo, sino un número de ciclos, que utiliza en el interior de un bucle. Pero debido al uso de un bucle contador, el tiempo de espera antes de dar el error depende de la frecuencia de reloj de la CPU, y es necesario adaptar los valores que usa el contador so pena de encontrar que una máquina rápida comienza a dar errores sin previo aviso.

El bucle del BIOS contiene el valor 4*65536 y en RAM se encuentra un factor que multiplica a este número para dar el número de ciclos. Por ejemplo, el valor 20 indica que se esperan durante 20*4*65536 ciclos antes de dar error. En un 486DX33 por ejemplo esto supone aproximadamente un segundo.

Como se dijo arriba, la funcion 0x00 se usa para enviar caracteres a la impresora. Aparte del número de función en AH, como siempre, colocamos el código ASCII del carácter en AL. Después de la llamada, como se ha dicho, se devuelve en AH el estado de la impresora. La función 0x01 sirve para iniciar la impresora y es una buena práctica llamarla antes de enviar datos por primera vez. Esta función se llama sólo con el número 0x01 en AH.

Finalmente, la función 0x02 tiene como única misión devolver el octeto de estado, poniéndolo en AH. Tampoco hay que suministrarle parámetro alguno.

6.5. Problemática de la programación de impresoras

6.5.1. Introducción

Desgraciadamente, nunca ha existido una norma aceptada universalmente en cuanto a un conjunto mínimo de órdenes que todas las impresoras pudiesen reconocer. De haberlo, la vida del programador sería mucho más sencilla. Aún cuando para impresoras de chorro de tinta y *laser* los lenguajes PCL y PostScript cubren casi cualquier impresora, en las impresoras matriciales el problema es verdaderamente agudo, pues cada fabricante dota a sus productos de un juego de códigos propio para realizar las funciones más habituales, como conmutación de modo carácter a modo gráfico, paso de espaciado fijo a proporcional, selección de distintos tamaños de letras, énfasis, etc.

Sin embargo, no todo está perdido. La mayoría de estas impresoras pueden emular a alguna de estas dos: IBM Proprinter XL24 y Epson. Por tanto, y

ya que dedicaremos un capítulo específico al lenguaje PostScript , en este presentaremos una descripción de las de órdenes XL24.

6.5.2. Las impresoras de la familia Proprinter XL24

Incluimos en este grupo tanto a la original como a cualquier otra que pueda emularla.

Códigos de control y secuencias de escape

Los códigos de control y las secuencias de escape son mensajes especiales que el programa de aplicación envía a la impresora. Estos mensajes modifican de alguna manera la forma en que la impresora ha imprimido hasta ese momento. Por ejemplo, si se desea subrayar una palabra será preciso enviar la secuencia de control adecuada, imprimir la palabra y restaurar el estado anterior.

Algunos de estos mensajes de control se envían a la impresora como caracteres únicos. Estos códigos de un solo carácter reciben el nombre de códigos de control. Por ejemplo, se usan códigos de control para conmutar a impresión comprimida, o para detener la impresión en un momento dado.

Las secuencias de escape son similares a los códigos de control ya que permiten modificar de alguna forma el funcionamiento de la impresora, pero a diferencia de los códigos de control, están formadas por secuencias de dos, tres o cuatro caracteres. Cada secuencia de escape, comienza con el carácter `ESC` (como era de esperar).

Explicación de los códigos y secuencias

Como se ha dicho, un código de control es un único octeto que se envía a la impresora y modifica su comportamiento. Cuando un código de control se encuentra precedido por `ESC`, este no tiene ningún efecto sobre el código de control. La impresora Proprinter dispone de tres páginas de códigos. La página 1 contiene códigos de control desde el caracter 0 (NULL) hasta el 32 (SP). Ademas, contiene códigos de control desde el caracter 128 (NULL) al 155 (ESC). El juego de caracteres 2 contiene códigos de control sólo desde el 0 al 32, y la página 3 no contiene caracteres de control.

Los códigos de control pueden colocarse en cualquier posición de la sucesión de datos. Pueden estar inmediatamente a continuación de los datos que se han imprimido o encontrarse inmediatamente antes de los mismos. Permanecerán en efecto hasta que se los cancele o hasta que se apague la impresora. Algunos códigos de control se cancelan en forma automática (por ejemplo, al final de la línea vigente) pero otros deberán ser cancelados por otro código de control o secuencia de escape.

La impresora XL24 Proprinter tienes tres modalidades de impresión. En el modo PD (Procesamiento de Datos) pueden verse los puntos individuales que forman los caracteres. Esta calidad se usa para borradores y trabajos de gran volumen. En modo PD la impresora original alcanza los 200 cps (caracteres por segundo). En la modalidad II (Impresion Intensificada) la velocidad baja a los 100 cps, se rellenan los espacios entre puntos adyaccentes y el resultado es más oscuro. En la tercera modalidad, CTC (Calidad Tipo Carta), se agregan puntos adicionales a los caracteres, aumentando la calidad del resultado.

Además, pueden definirse hasta 256 caracteres de usuario, que se almacenan en la propia impresora, y pueden imprimirse gráficos de puntos direccionables. Es decir, pueden imprimirse gráficos arbitrarios compuestos por puntos que se colocan en cualquier posición del papel.

Finalmente, pueden combinarse dos modalidades de impresión distintas para crear una nueva. Por ejemplo, doble anchura enfatizada, doble altura condensada, etc.

Las órdenes y secuencias de escape pueden enviarse a la impresora desde un .bat, un programa de aplicación o desde la misma línea de órdenes. Por ejemplo:

```
echo _G > prn
```

inicia el modo CTC ('_' representa al carácter ASCII 155), mientras que

```
echo _H > prn
```

lo interrumpe. Finalmente, cabe decir que la impresora IBM Proprinter XL24 permite redefinir los caracteres, proporcionando secuencias de escape para efectuar la definición, cargar juegos de caracteres completamente distintos, imprimir un carácter en concreto de un juego dado, etc. Remito al lector al manual técnico de IBM para los detalles. Allí encontrará también tablas completas de comandos, y ejemplos de programación.

6.6. Instalación y administración de impresoras en UNIX

6.6.1. Introducción

UNIX dispone de un subsistema muy potente para el control de impresión. De acuerdo con la filosofía UNIX de disponer de varias herramientas sencillas que realizan tareas específicas y que pueden combinarse entre sí para efectuar trabajos más complejos, el subsistema de impresión, denominado "subsistema lp", es de una flexibilidad notable. Permite configuraciones múltiples, desde el uso de una impresora por varios usuarios hasta la impresión remota, pasando por la configuración para servidor de impresión y la administración de varias colas destinadas a distintas impresoras. De hecho, no se usa otro software distinto para impresión bajo UNIX.

6.6.2. Orden lp

La impresión se invoca mediante la orden `lp`, que puede usarse como terminación de un cauce o con argumentos:

```
# cat /etc/documento | lp
# lp /etc/documento
```

`lp` coloca el trabajo en cola y vuelve al *shell*. El trabajo puede ser impreso justo a continuación o puede llevar un tiempo, dependiendo del estado de la cola. Por eso es útil la opcion `-m` (mail), que comunica al usuario por correo electrónico la finalización del trabajo. La comunicación también puede hacerse mediante un mensaje en el terminal, usando la opcion `-w`. Otra opción interesante es `-nx`, donde `x` es el número de copias deseadas. Por ejemplo:

```
# lp -m -n2 /etc/documento
```

Cuando se comparte una impresora, según el tipo de salida, al usuario le puede resultar difícil identificar su trabajo, y por eso es útil la opcion `-t`, que permite añadir un *página insignia* que identifique cada trabajo. Por ejemplo:

```
# lp -m -n2 -t"Pertenezco a $LOGNAME" /etc/documento
```

Después de que `lp` ha colocado el trabajo en cola, devuelve en el terminal un identificador para ese trabajo, que generalmente consta del nombre de la impresora seguido de un número. Por ejemplo:

```
# lp /etc/documento
# Request id is IBMXL24-12
```

Si se dispone de varias impresoras, es posible indicar la impresora en concreto adonde irán los trabajos:

```
# lp -d IBMXL24 fichero1 fichero2
```

y por otra parte, si se ha iniciado un trabajo, puede cancelarse mediante la orden `cancel`, de esta forma:

```
# cancel IBMXL24-12
```

produce la cancelación del trabajo 12 de la impresora IBMXL24, mientras que

```
# cancel IBMXL24
```

produce la cancelación de todos los trabajos enviados a esa impresora.

6.6.3. Filtros de impresión

Una impresora puede estar configurada para trabajar con distintos tipos de ficheros. Por ejemplo, ASCII, ASCII extendido, PostScript, etc. Cuando se llama a `lp`, éste examina al fichero para tratar de determinar su tipo. Si no lo consigue, supondrá que se trata de un fichero simple que puede ser impreso directamente en una impresora sencilla. Si determina que es un fichero .ps, por ejemplo, y la impresora destino acepta ese formato, lo pondrá directamente en cola, pero si la impresora destino no acepta el formato, `lp` invocará a un filtro que haga las conversiones necesarias. En cualquier caso, `-T` permite especificar el tipo de fichero:

```
# lp -T postscript fichero.ps
```

6.6.4. Estado del subsistema lp

La orden `lpstat` proporciona información del estado del subsistema lp. Invocada sin opciones, `lpstat` devuelve el estado de la cola del usuario:

```
# lpstat
IBMXL24-12 gil 14702 Apr 20 10:02
    |       |    |      |     |
    |       |    |      |     +> hora
    |       |    |      +> fecha
    |       |    +> tama#o
    |       +> id   +> usuario
```

Si el trabajo ya esta imprimiéndose, se informará de ello:

```
# lpstat
IBMXL24-12 gil 14702 Apr 20 10:02 on IBMXL24
```

`lpstat` admite muchas opciones útiles, algunas de las cuales son:

Opción	Función
-d	indica la impresora por defecto
-r	indica si el subsistema lp esta funcionando o no
-p X	informa sobre la impresora X
-p X -l	informa in extenso sobre la impresora X

6.6.5. Acciones administrativas

La conexión de una impresora requiere de unas acciones físicas y de unas acciones lógicas. Entre las primeras se encuentran la conexión física a un puerto serie o paralelo, usando los cables adecuados y la configuración de la impresora, según las especificaciones del fabricante. Las acciones lógicas se realizan a través de los ficheros de dispositivo UNIX, que son los siguientes:

Dispositivo DOS	Dispositivo UNIX
LPT1	/dev/lp
	/dev/lp0
LPT2	/dev/lp1
LPT3	/dev/lp2
COM1	/dev/tty00s
	/dev/term/tty00s
COM2	/dev/tty01s
	/dev/term/tty01s
COM3	/dev/tty02s
	/dev/term/tty02s
COM4	/dev/tty03s
	/dev/term/tty03s

Para instalar una impresora es preciso en primer lugar verificar su correcta conexión. En segundo lugar debe examinarse el *guión de interfaz* para asegurarse de que hace lo que se supone que debe hacer. El sistema lp funciona pasando el fichero de salida a través de un programa de interfaz, que es generalmente un guión *shell*. Este guión prepara la página insignia, configura el puerto usando `stty` y escribe los datos en el fichero de dispositivo adecuado. En tercer lugar se especificarán unos valores por omisión que serán usados por lp cuando el usuario no especifique otros. Adicionalmente pueden establecerse una serie de filtros de conversión. Finalmente, se habilitará la impresora.

Repasaremos con algún detalle cada uno de estos pasos. Puede verificarse la correcta conexión de la impresora escribiendo en el archivo de dispositivo al que se piensa que la impresora está conectada. Por ejemplo:

```
# cat /etc/prueba > /dev/lp0
```

Si no aparece nada en la impresora, verifíquense los cables y los interruptores. Si aparece una salida que difiere de la entrada, (por ejemplo, caracteres cambiados) compruébese la configuración de la impresora. Cuando la impresora esta preparada para PostScript u otro lenguaje de descripción de páginas, es necesario enviar un programa que pueda ser interpretado por la impresora. Por ejemplo, el fichero prueba.ps:

```
# cat prueba.ps
/Times-Roman findfont 14 scalefont setfont
300 400 moveto
```

```
(Hola impresora!) show
showpage
# cat prueba.ps > /dev/tty00s
```

debe producir la impresión de "Hola impresora!".

Los modelos de interfaz se encuentran en `/usr/lib/lp/model`. Estos modelos son responsables de definir los atributos de la conexión, como velocidad y control de flujo, formateo de la página insignia, preparación de múltiples copias, etc. El modelo *estandard* funciona con casi todas las impresoras, y si se precisa escribir un guión nuevo, es una buena idea copiar el primero en un directorio distinto y modificar aquello que sea necesario.

La herramienta de configuración de la impresora es `/usr/sbin/lpadmin`, y se usará una vez comprobada la correcta conexión y configuración de la misma, y cuando el subsistema lp se encuentre activo. Supongamos que se quiere dar de alta una nueva impresora. Se le puede dar el nombre que se desee, pero es preferible que sea un nombre descriptivo. Sea HP para los ejemplos que siguen. `lpadmin` puede hacer referencia a HP escribiendo:

```
#lpadmin -p HP
```

Una linea típica para añadir una impresora es la siguiente:

```
# lpadmin -p HP -m standard -v /dev/tty00s
            ----- ----------- --------------
              |        |             |
              |        |             |
              |        |             |
              |        |             |
              |        |             +> fichero dispositivo
              |        +> guion a usar
              +> nombre
```

Si se va a usar un guión distinto de los ofrecidos por el sistema, hay que usar `-i` en lugar de `-m`.

No siempre es fácil determinar el tipo de impresora. En la base de datos *terminfo* se encuentra la información requerida, y puede consultarse mediante el programa `infocmp`. En terminfo se encuentra el nombre de la impresora (o una abreviatura más o menos fácilmente reconocible) junto con información sobre cómo genera saltos de página, número de caracteres por línea, número de líneas por página, etc. Podemos especificar el tipo mediante:

```
# lpadmin -p HP -T laserjet
```

En cuanto al contenido de los ficheros que se enviarán a la impresora, se especifican con `-I`:

```
# lpadmin -p HP -I postscript
```

Si no se especifica ningún tipo, se supone que son ficheros simples.

Como se ha dicho antes, los valores de funcionamiento por defecto se toman de *terminfo*, pero pueden modificarse mediante la opcion `-o`. Véase bibliografía. La eliminación de una impresora del sistema es mucho más sencilla:

```
# lpadmin -x HP
```

Ahora bien, no es suficiente lo dicho hasta ahora para poder imprimir. Es necesario distinguir entre las acciones administrativas que informan al sistema de la existencia de una impresora, y el que esta impresora esté *capacitada* o no para imprimir. Por ejemplo, si accidentalmente se desenchufa la impresora, el sistema automáticamente la discapacita para evitar el que se le envíen trabajos. Puede capacitarse o discapacitarse una impresora explícitamente mediante las órdenes `/usr/sbin/accept` y `/usr/sbin/reject`. Con la primera, la impresora queda dispuesta ya para imprimir; con la segunda, `lp` rechazará incorporar trabajos a la cola de la impresora, pero sin retirarla del sistema.

6.6.6. Otras acciones

El subsistema lp admite muchas más funciones: pueden trasladarse trabajos de una cola de impresión a otra, puede especificarse que un trabajo se imprima sobre un formulario determinado, pueden cambiarse los juegos de caracteres de una impresora, puede configurarse una máquina como servidor de impresión, pueden enviarse solicitudes de impresión via ucpp, etc. Por último, recuérdese que lp puede trabajar al final de una línea de cauce que

contenga órdenes para dar formato a un texto, desde el sencillo pero efectivo `pr` hasta el impresionante paquete LaTeX.

6.7. Particularidades de la impresión en Linux

6.7.1. Introducción

Esta sección repasa específicamente la impresión bajo Linux, y se basa en la documentación en línea suministrada por todas las distribuciones, a la que remito al lector si desea ampliar la información.

Linux permite usar muchas impresoras, pero existe un problema con aquellas impresoras etiquetadas con las palabras *for Windows*. En este tipo de impresoras muchas de las acciones que efectuaba el procesador de la impresora ahora las efectúa la CPU del equipo, siguiendo las órdenes del controlador de impresora suministrado por el fabricante, sólo disponible para este S.O.. Por eso, a la hora de comprar una impresora, es preferible elegir una independiente del Sistema Operativo, ya que no parece nada razonable que una vez comprada una impresora *para Windows* no se pueda emplear con un sistema operativo más avanzado, como Linux, FreeBSD o, en general, cualquier miembro de la familia UNIX.

Una opción es consultar la ayuda de `gs` mediante la orden:

```
$ gs -h | more
```

y comprobar para qué familias de impresoras está disponible el intérprete PostScript (ver más adelante en esta sección).

6.7.2. Impresión usando lpr

La forma más simple de imprimir bajo Linux es escribiendo directamente en el archivo de dispositivo, aunque para eso es preciso tener privilegios administrativos:

```
# cat tesis.txt > /dev/lp
```

Puesto que sólo 'root' y usuarios de su grupo pueden escribir en `/dev/lp`, el resto ha de usar órdenes como `lpr`, `lprm` y `lpq`.

La orden `lpr` realiza el trabajo inicial necesario y pasa el control a `lpd`, el demonio de impresión, que es quien controla la impresora. Cuando `lpr` se ejecuta, primero copia el fichero a un directorio llamado `/spool`, donde permanece hasta que `lpd` lo imprime. Cuando `lpr` le indica a `lpd` que tiene un archivo que imprimir, `lpd` hace una copia de sí mismo (mediante fork()). Esta copia se encarga de la impresión, mientras que el original permanece a la espera de nuevos envios. La sintaxis de `lpr` es familiar:

```
$ lpr [opcion] [fichero]
```

Si no se especifica `[fichero]`, `lpr` espera la entrada desde la entrada predeterminada, usualmente el teclado, o la salida de otra orden, lo que permite usar cauces, como en:

```
$ pr -l60 tesis.txt | lpr
```

Algunos de los argumentos más comunes usados con `lpr` son `-Pprinter`, para especificar el tipo de impresora, `-h`, que suprime la página que identifica al propietario del trabajo (esta página puede ser necesaria si muchos usuarios estan usando una impresora en red, por ejemplo, y ayuda a que cada uno de ellos encuentre rápidamente su trabajo, una vez impreso), `-s`, que crea un enlace simbólico en lugar de copiar el archivo en `/spool` y `#numero`, que indica el número de copias a efectuar.

Por su parte, `lpq` sirve para visualizar la cola de impresión, lo que puede ser útil en algunas ocasiones. Por ejemplo, para decidir si se cancela un trabajo o no según que haya o no comenzado a imprimirse. Para borrar un trabajo, la orden es `lprm`. En su versión más simple:

```
$ lprm -
```

borra todos los trabajos del usuario que la invoca. Para cancelar un trabajo en concreto, se le indica el número de trabajo, por ejemplo:

```
$ lprm 17
```

donde el número '17' ha sido mostrado previamente por `lpq`

6.7.3. El programa lpc

El programa `lpc` se usa para controlar las impresoras y el demonio `lpd`, reordenar las colas de impresión y obtener el estado de una o todas las impresoras. Si se invoca como:

```
$ lpc
```

entra por defecto en un modo interactivo. Algunos de los comandos habituales son:

1. `disable [opcion]` impide la introducción de nuevos trabajos.
2. `down [opcion]` deshabilita cualquier impresión
3. `enable [opcion]` habilita la introducción de nuevos trabajos en la cola
4. `quit` abandona lpc
5. `restart [opcion]` rearranca `lpd` para la impresora indicada
6. `status [opcion]` devuelve el estado de la impresora
7. `up [opcion]` habilita 'todo' y arranca un nuevo `lpd`

6.7.4. Impresión de tipos concretos de archivos

Sería ideal que sólo existiese un formato para los archivos de impresora, digamos PostScript. Pero no es así. En cualquier caso, imprimir PostScript (PS) en una impresora PS es tan simple como hacer:

```
$ cat tesis.ps | lpr
```

ya que toda la interpretación del archivo corre a cargo de la impresora. Pero cuando no se dispone de impresora PS, Linux proporciona programas para convertir un archivo .ps en un archivo con órdenes para la impresora de que dispongamos. De modo que podemos restringirnos casi exclusivamente a archivos PS como formato para nuestros documentos, y usar después un filtro para trasladar estos archivos a otros formatos.

El más extendido de estos programas es GhostScript (`gs` en adelante). Usualmente, se usa `gs` para visualizar en pantalla documentos PS, pasando desapercibido el hecho de que la salida a terminal gráfico es una de las muchas de que dispone el programa.

Por defecto, `gs` toma un .ps y trás interpretarlo lo envía a un terminal x11, donde lo visualizamos. Pero este comportamiento puede cambiarse mediante la opción `-sDEVICE=nombre`, donde `nombre` puede ser alguno de los siguientes:

```
x11 x11alpha x11cmyk x11gray2 x11mono ap3250 imagen iwhi
iwlo iwlq la50 la70 la75 la75plus lbp8 ln03 lj250 lj4dith
lp2563 m8510 necp6 oce9050 r4081 sj48 st800 stcolor t4693d2
t4693d4 t4693d8 tek4696 xes deskjet djet500 djet500c dnj650c
laserjet ljetplus ljet2p ljet3 ljet3d ljet4 cdeskjet
cdjcolor cdjmono cdj500 cdj550 paintjet pj pjxl pjxl300
uniprint bj10e bj200 bjc600 bjc800 epson eps9mid eps9high
epsonc ibmpro jetp3852 dfaxhigh dfaxlow faxg3 faxg32d faxg4
cp50 pcxmono pcxgray pcx16 pcx256 pcx24b pcxcmyk pbm pbmraw
pgm pgmraw pgnm pgnmraw pnm pnmraw ppm ppmraw tiffcrle
tiffg3 tiffg32d tiffg4 tifflzw tiffpack tiff12nc tiff24nc
psmono psgray bit bitrgb bitcmyk pngmono pnggray png16
png256 png16m jpeg jpeggray pdfwrite pswrite epswrite
pxlmono pxlcolor nullpage
```

Por ejemplo:

```
gs -sDEVICE=hpljii tesis.ps
```

toma `tesis.ps`, lo interpreta y lo transforma al conjunto de órdenes que puede entender una impresora HP LaserJet II. Puede entonces redirigirse la salida directamente a nuestra impresora mediante `-sOutputFile=lpr`. Por ejemplo:

```
gs -sDEVICE=hpljii -sOutputFile=lpr tesis.ps
```

Por supuesto, los procesadores de texto profesionales, como TEX, LATEX o troff pueden generar salida PostScript. En particular, las páginas *man* contienen archivos fuente troff. Podemos limpiar estos archivos de comandos troff mediante el programa `col` y pasarlos después a la impresora o a un archivo. Por ejemplo, para imprimir las páginas de manual de `gs`:

```
man gs | col -b | lpr
```

6.7.5. Los filtros mágicos y el archivo /etc/printcap

Un filtro es un programa que toma un archivo de entrada y produce una salida después de haber efectuado algún tipo de operación sobre el archivo. Un filtro 'mágico' es aquel que deduce por medio de algún algoritmo heurístico el formato del archivo y toma la decisiones apropiadas para que el archivo se imprima correctamente. Dos de estos programas son `APSFilter` y `MagicFilter`. Por ejemplo, si queremos imprimir un archivo .ps, APSFilter explorará el archivo, deducirá que es de tipo PostScript, usará GhostScript para convertirlo al formato de nuestra impresora y enviará la salida a lpr.

Un problema común cuando se estan imprimiendo archivos PS consiste en el mensaje de error 'archivo demasiado grande'. Ocurre que en el archivo `/etc/printcap` (este archivo es similar al `/etc/termcap`, y contiene las descripciones de las impresoras que pueda usar el sistema) contiene una opción que previene la impresión de archivos demasiado grandes, limitando el tamaño a 1000 bloques de disco, que en un archivo PS normal pueden ser del orden de una docena de páginas. Esta es una característica de seguridad que trata de prevenir que se llene la partición que contiene a `/spool`. Para desactivar esta limitación, añádase el comando `mx=0` en `/etc/printcap`. Otra opción consiste en hacer que lpr cree un enlace simbólico entre `/spool` y el archivo a imprimir, pero entonces habrá que invocar siempre a lpr con la opción `-s`. De todas formas, si dispones de una distribución de la familia RedHat, puedes usar el programa `printtool`. Este programa es una interfaz que crea descripciones en `/etc/printcap`. Puedes crear algunas descripciones y luego examinar el resultado, para ver como se traducen a órdenes las opciones que hayas elegido.

Capítulo 7

El lenguaje PostScript

7.1. Introducción

PostScript es un lenguaje de programación diseñado para la descripción de páginas. Contiene una gran variedad de operadores gráficos que pueden ser combinados en procedimientos y funciones. Este tema es un resumen del libro "PostScript language. Tutorial and Cookbook", de Adobe System Inc., y se limita a dar una introducción rápida, sin entrar en temas más específicos, como la gestión y definición de fuentes y la integración de gráficos con texto.

Los programas PostScript son interpretados, y pueden ser generados de tres formas distintas: directamente por un programador, generados por un programa de aplicación o introducidos en aquellas impresoras que soportan un modo interactivo. Los operadores gráficos de que dispone PostScript permiten trabajar con tres tipos de objetos:

1. Texto, que puede presentarse en una gran variedad de fuentes, situado en cualquier punto de la página, con cualquier orientación y a cualquier escala.

2. Figuras geométricas, formadas por líneas rectas, así como curvas de cualquier tamaño, orientación y grosor, y rellenos de cualquier forma, tamaño y color, con una gran variedad de tramas.

3. Imágenes procedentes de programas de aplicación o fotografias digitalizadas, que pueden colocarse en cualquier sitio, con cualquier orientación y escala.

Para PostScript, la composición de una página se asienta en tres conceptos fundamentales:

- La página actual. Es una página virtual en donde PostScript dibuja. Al comienzo de un programa, la página actual se encuentra vacía. Los operadores PostScript rellenan la página, que una vez compuesta es enviada a la impresora.
- La trayectoria actual. Es el conjunto de puntos, conectados o no, líneas y curvas, con la descripción de formas y tamaños. Este conjunto puede definirse independientemente de la resolución de la impresora.
- La zona activa: Es el contorno del área que va a ser dibujada. Inicialmente, la zona activa coincide con el tamaño por defecto del papel para la impresora. La zona activa puede ser dimensionada a cualquier tamaño y forma. Si un operador trata de dibujar fuera de la zona actual, es ignorado.

La posición de un punto sobre el papel se define por un par de coordenadas (x,y). Ocurre que cada impresora tiene su propia escala y orientación, de forma que se puede hablar de un .espacio de impresoraτ de un .espacio de usuario". Este último es definido por el usuario y transformado automáticamente mediante una transformación lineal en cada eje al espacio de la impresora antes de la impresión. El usuario puede cambiar el origen a cualquier punto, alterar la escala independientemente para cada eje y rotar los ejes.

7.2. PostScript como lenguaje de programación

PostScript es un lenguaje de programación, y como tal comparte características con otros lenguajes, aunque se encuentra cercano a Forth. Algunos aspectos de PostScript son:

- La Pila. PostScript usa una pila LIFO para colocar los datos con los que trabaja.
- Uso de notación postfija. Los operadores toman los operandos de la pila, de modo que estos han de ser colocados en primer lugar.
- Soporta muchos tipos de datos, comunes a otros lenguajes de programación, como reales, enteros, cadenas, booleanos y arrays.

- Soporta funciones, que una vez definidas pueden usarse como cualquier otro operador. Además, descripciones parciales de una página pueden usarse tal cual en otras páginas, o transformadas mediante escalado, traslación o rotación para ser usadas en otras páginas.
- Los programas PostScript están escritos en caracteres ASCII imprimibles, y por tanto pueden ser editados prácticamente por cualquier editor y procesador de textos.

7.3. La pila. Aritmética

Fundamentalmente, hay dos formas en que un lenguaje de programación puede manipular los datos: Pueden asignarse a variables y ser direccionados por los nombres de las variables o pueden manipularse directamente usando una pila. PostScript soporta los dos métodos.

Cuando el intérprete encuentra objetos, como números o cadenas de caracteres, en una línea del código fuente, los coloca directamente en la pila. Por ejemplo, si una línea de un programa PostScript es:

```
10  -.98  12.23
```

el intérprete coloca en la pila primero el 10, luego -.98 y finalmente 12.23. En una línea pueden encontrarse uno o muchos objetos, que estarán separados por espacios, tabuladores o caracteres de nueva línea.

Otro tipo de objetos que el intérprete puede encontrar son los operadores. Cuando encuentra una palabra, el intérprete busca en un diccionario interno para comprobar si es un operador. Si lo es, realiza las operaciones apropiadas. Los operadores toman los datos de la pila, eliminándolos de la misma y apilando en su lugar los resultados.

A continuación se presentan algunos de los operadores más comunes, ilustrando la forma en que actúan:

```
2 2 add     ->      4
3 2 sub     ->      1
13 8 div    ->      1.625
25 3 idiv   ->      8
12 10 mod   ->      2
6 8 mul     ->      48
-27 neg     ->      27
```

(Al lector familiarizado con las calculadoras científicas HP, le resultará familiar esta forma de proceder) Expresiones más complejas se realizan de forma similar. Algunos ejemplos serán suficientes para comprender la notación postfija:

```
6 + (3/8)   ->    3 8 div 6 add
8 - (7*3)   ->    8 7 3 mul sub
```

Otros operadores de uso común son: `dup`, que duplica el último elemento de la pila; `exchg`, que intercambia los dos últimos elementos y `pop`, que elimina el último elemento de la pila.

7.4. Primeros programas en PostScript

Presentaremos algunos programas sencillos para ilustrar tanto la estructura y aspecto de los fuentes como los operadores gráficos más comunes.

```
% primer programa en PostScript
newpath
144 72 moveto
144 132 lineto
stroke
showpage
```

El operador `newpath` inicia la trayectoria actual. Para iniciar una nueva trayectoria, hemos de colocarnos en el punto de la página que deseemos. En este caso, el punto elegido es (144,72), que se convierte en el punto actual. El sistema de referencia por defecto tiene su origen en la esquina inferior izquierda, con el eje X hacia la derecha y el eje Y hacia arriba. La unidad de medida son 0.3528 milimetros (1/72 pulgadas). La línea `144 132 lineto` mueve el cursor sobre la página virtual a la posición (144,132). Nótese que aún no se dibuja la línea que conecta (144,72) con (144,132), sino sólamente este segundo punto es asociado a la trayectoria. El operador `stroke` fuerza a que la trayectoria definida sea dibujada sobre la página virtual, que finalmente es impresa mediante el operador `showpage`. Obsérvese que los comentarios comienzan con `%`.

```
% segundo programa PostScript
newpath
72 360 moveto
```

```
144 72 rlineto
144 432 moveto
0 -216 rlineto
stroke
showpage
```

Comenzamos definiendo una nueva trayectoria y colocando el cursor. La siguiente línea presenta el operador `rlineto`, que es similar a `lineto` salvo por el hecho de que los operandos ahora representan desplazamientos respecto de la posición actual del cursor. Las siguientes dos líneas forman una recta que intersecta a la primera. Finalmente, dibujamos la trayectoria con `stroke` y mostramos la página. Lo notable de este programa es que la trayectoria no tiene que ser continua, sino que puede estar compuesta por cualquier número de trozos continuos.

```
% tercer programa en PostScript
newpath
270 360 moveto
0 72 rlineto
72 0 rlineto
0 -72 rlineto
-72 0 rlineto
4 setlinewidth
stroke
showpage
```

Este programa dibuja un recuadro cuyos lados tienen un grosor de 4 veces 0.3528 milímetros. Pero, debido a este grosor apreciable, la esquina inferior izquierda no queda bien dibujada. Para solucionar este problema, se usa el operador `closepath`:

```
% cuarto programa en PostScript
newpath
270 360 moveto
0 72 rlineto
72 0 rlineto
0 -72 rlineto
closepath
4 setlinewidth
stroke
showpage
```

```
% quinto programa en PostScript
newpath
270 360 moveto
0 72 rlineto
72 0 rlineto
0 -72 rlineto
closepath
fill
showpage
```

Aquí, en lugar de llamar a `stroke` para que dibuje el camino, llamamos a `fill`, para que rellene el polígono formado. Por defecto, el polígono se rellena en negro, pero puede definirse el grado de gris que se desee mediante el operador `setgray`. El argumento de este operador es un número que puede tomar valores entre 0 (negro) y 1 (blanco). Por otra parte, las figuras son opacas, de forma que cualquier cosa que se dibuje sobreescribe lo que hubiese debajo. El siguiente programa dibuja tres cuadrados que se superponen, cada uno con un grado de gris:

```
% sexto programa en PostScript
newpath
252 324 moveto
0 72 rlineto              % este es el resultado
72 0 rlineto              %
0 -72 rlineto             %
closepath                 %       ooooooooooo
fill                      %     xxooooooooooo
newpath                   %   ZZxxoooooooooooo
270 360 moveto            %   ZZxxoooooooooooo
0 72 rlineto              %   ZZxxoooooooooooo
72 0 rlineto              %   ZZxxoooooooooooo  <- claro
closepath                 %   ZZxxxxxxxxxxxx
0.4 setgray               %   ZZZZZZZZZZZZ    <- oscuro
fill                      %
newpath                   %
288 396 moveto
0 72 rlineto
72 0 rlineto
closepath
0.8 setgray
fill showpage
```

7.5. Procedimientos y variables

7.5.1. Diccionarios

Un diccionario es una lista que asocia pares de objetos. Bajo esta definición amplia, PostScript contiene dos diccionarios: el del sistema y el del usuario. El diccionario del sistema asocia a cada operador predefinido una determinada acción. El diccionario del usuario asocia nombres con las variables y procedimientos definidos por un programa. Cuando el intérprete encuentra un nombre, busca primero en el diccionario del usuario, y después en el del sistema. Si encuentra el nombre, realiza las acciones asociadas a ese nombre. Los diccionarios a su vez se almacenan en una pila, con el diccionario del sistema en la base y el diccionario del usuario en la cima. Un programa puede crear nuevos diccionarios, que serán apilados sobre los ya existentes. Aquel que se encuentra en la cima de la pila de diccionarios, es el diccionario actual.

7.5.2. Definición de variables

Para definir una variable, se usa el operador `def`, con la sintaxis siguiente:

```
/nombre valor def
```

La barra indica al intérprete que no trate de buscar el valor de `nombre`, sino que tome su valor literal y lo ponga en la pila. A continuación, `valor` es apilado y, finalmente, `def` toma ambos valores y los añade al diccionario actual. El valor de la variable puede redefinirse más tarde, como muestran los siguientes ejemplos:

```
/rosa 72 def              % rosa = 72
  .
  .
/rosa rosa 1 add def      % rosa = rosa + 1
/rosa 1000 def            % rosa = 1000
```

7.5.3. Definición de procedimientos

En PostScript, un procedimiento es un conjunto de operaciones agrupadas con un nombre común. Los procedimientos se definen igual que las variables, con el conjunto de operaciones a realizar entre llaves. Por ejemplo:

```
/Proc1 {72 mul} def
```

es un procedimiento que multiplica el valor de la pila por 72. De esta forma, `5 72 mul` es idéntico a `5 Proc1` , y ambas líneas dejan en la pila el valor 360. Repetiremos el programa que dibuja tres recuadros solapados.

```
% definir procedimiento del recuadro
/recuadro
  {  72 0 rlineto
     0 72 rlineto
     -72 0 rlineto
     closepath } def

% comienza el programa
newpath
252 324 moveto
recuadro
0 setgray
fill
newpath
270 360 moveto
recuadro
0.4 setgray
fill
newpath
288 396 moveto
recuadro
0.8 setgray
fill
showpage
```

7.6. Texto

PostScript es muy potente en cuanto a las posibilidades para imprimir texto. Puede definirse una fuente, y el texto que use esa fuente puede escalarse, trasladarse, rotarse o cualquier combinación de estas operaciones.

En PostScript, las fuentes son descritas geométricamente, de forma que puedan imprimirse a cualquier escala. Antes de imprimir texto, es necesario especificar la fuente, proceso que requiere tres pasos : en primer lugar, es necesario encontrar la descripción de la fuente, información que se encuentra

en un diccionario. En segundo lugar, hay que escalar la fuente al tamaño deseado. Finalmente, la fuente escalada se establece como fuente actual. Como ilustración, el siguiente programa escribe "PostScript.[en] Times-Roman de 15 puntos:

```
/Times-Roman findfont
15 scalefont
setfont
72 200 moveto
(PostScript) show
showpage
```

En la primera línea, ponemos la cadena `Times-Roman` en la pila, y llamamos a `findfont`, que busca el nombre en el diccionario llamado *FontDirectory* y coloca la fuente apropiada en la pila. La información devuelta por `findfont` contiene la descripción de la fuente Times-Roman de tamaño 1, de manera que hay que escalarla a tamaño 15 mediante la tercera línea. Una vez establecida la fuente con `setfont`, se mueve el cursor al punto deseado, se pone el texto entre paréntesis para indicar que es una cadena lo que ha de apilarse y se llama al operador `show`, que imprime la cadena a partir de la posición del cursor.

Para PostScript, no hay diferencia entre texto y gráficos, de forma que ambos pueden combinarse como se desee. Cuando se va a imprimir un texto, es conveniente saber que tamaño va a ocupar. Para esto usamos el operador `stringwidth`. Este operador toma una cadena de la pila y devuelve los incrementos x e y, en este orden, que experimentaría el punto actual si la cadena fuese impresa.

7.7. Sistemas de coordenadas

Como ya se ha dicho, existe un *espacio de dispositivo* y un *espacio de usuario*, cada uno con su propio sistema de referencia. El de usuario tiene su origen en la esquina inferior izquierda de la página virtual, con el eje X hacia la derecha y el Y hacia arriba. El origen del sistema de referencia de usuario (SRU) puede trasladarse mediante el operador `translate`, por ejemplo:

```
20 80 translate
```

coloca el origen del SRU en (20,80). Igualmente, puede rotarse el SRU un numero de grados que se especifique en la pila. Así:

```
60 rotate
```

gira el SRU en 60 grados en sentido contrario a las agujas del reloj. En lugar de escribir código para trasladar el recuadro a cada nueva posición, podemos simplemente trasladar el origen, ahorrando código y haciendo más legible el que se escriba.

La otra operación que puede hacerse sobre el SRU es el escalado, que permite cambiar el tamaño de las unidades que usa PostScript , independientemente para cada eje. Por ejemplo:

```
3 3 scale
```

triplica el tamaño de la unidad de medida, y por tanto el tamaño de los objetos que se dibujen.

```
1 2 scale
```

duplica la unidad sobre el eje Y, haciendo que los objetos aparezcan alargados. Análogamente,

```
2 1 scale
```

los hace aparecer achatados.

7.8. Estructuras de control del programa

7.8.1. Comparaciones

PostScript contiene un conjunto completo de operadores de comparación:

```
eq  ->  =      ne  ->  !=
gt  ->  >      lt  ->  <
ge  ->  >=     le  ->  <=
```

El resultado booleano de una comparación puede a su vez usarse como argumento para los operadores lógicos, también incluidos en el lenguaje:

```
not
and
or
xor
```

7.8.2. Condicionales

El operador `if` toma de la pila un booleano y una cadena, y realiza las operaciones de la cadena siempre que el booleano sea *true*. Consideremos por ejemplo el siguiente fragmento de código:

```
/ver_si_fin_de_linea
{  currentpoint pop    % retira coordenada Y
  612 gt
  { 0 -12 translate 0 0 moveto } if
} def
```

El operador `ifelse` requiere tres objetos en la pila: un booleano y dos *arrays* ejecutables. El primero se ejecutará si el booleano es *true*, y el segundo en caso contrario.

7.8.3. Bucles

El operador `repeat` toma dos argumentos de la pila, un número que indica las repeticiones, y un conjunto de operaciones entre llaves, operaciones que pueden incluir llamadas a procedimientos.

El operador `for` toma cuatro argumentos: el valor de inicio del contador del bucle, el incremento en cada iteración, el valor final y el procedimiento que se va a repetir. La siguiente línea de código hace que la letra *k* se imprima cada doce unidades a lo largo de la página:

```
0 12 600 {0 moveto (k) show} for
```

los argumentos numéricos de `for` no necesitan ser enteros.

El operador `loop` se usa para aquellos bucles que haya que repetir un número indefinido de veces, hasta que se cumpla cierta condición. El operador `loop` toma un array ejecutable, y lo ejecuta hasta que un `exit` dentro del procedimiento cause la salida del `loop` más interno. Por ejemplo:

```
{ (parva bellua capillata) show } loop
```

repite la cadena *parva bellua capillata* indefinidamente.

Aunque no es una estructa de control, la recursión es una potente característica de PostScript, que aplicada a gráficos permite dibujar fácilmente complejas estructuras fractales. Consúltese la bibliografía para los detalles.

7.9. Vectores

En PostScript, los vectores [1] son colecciones unidimensionales de objetos , numerados de 0 a $n-1$, siendo n el número de tales objetos en el vector. PostScript destaca de otros lenguajes de programación en que los elementos de un vector no tienen que ser del mismo tipo. Por ejemplo, un vector puede contener números, cadenas, procedimientos, diccionarios... todo al mismo tiempo. También puede contener operadores, que se ejecutan cuando se define el vector, por ejemplo:

```
[ 16 (doce) 8 ]
```

contiene dos números y una cadena, mientras que:

```
[ (hola) 6 12 add ]
```

es un vector que contiene una cadena y un número. Los vectores también pueden definirse mediante el operador `array`, cuya sintaxis es:

```
<numero_elementos> array
```

Por ejemplo, `10 array` coloca en la pila un vector de diez elementos, a los que se asigna el valor `null`.

Los operador `put` y `get` sirven para manipular los objetos de un vector. El primero toma tres argumentos: un vector, un índice y un objeto, colocando éste en la posición indicada por `indice` dentro del vector. Por ejemplo:

```
/Mi-array 10 array def
Mi-aray 8 (vaca) put
```

coloca la cadena `vaca` en la posición 9 del vector `Mi-array`, definido en la primera línea. `get` toma como argumento un vector y un índice, tomando del primero el elemento indicado por el segundo y colocándolo en la pila. Así:

```
[ 2 5 9 ] 1 get
```

coloca 5 en la pila.

[1] ¿ Por qué no usar la palabra vector en lugar del anglicismo *array* ?

Muy frecuentemente ha de realizarse una misma operación sobre todos los elementos de un vector, para lo que es especialmente adecuado el operador `forall`, que toma como operandos un vector y un procedimiento, que es aplicado a cada uno de los objetos.

Finalmente, describiremos los operadores `aload` y `astore`. El primero toma un vector como argumento, y coloca en la pila cada uno de sus elementos, y después al propio vector. Así, la operación:

```
[ 1 2 3 ] aload
```

tiene como resultado el almacenamiento en la pila de la serie

```
1 2 3 [ 1 2 3 ]
```

Por el contrario, `astore` toma una serie de elementos de la pila y los almacena en un vector, colocándolo en la pila:

```
(a) (b) (c) (d) 4 array astore
```

tiene como resultado colocar en la pila

```
[ (a) (b) (c) (d) ]
```

7.10. Ejercicio

Este es un pequeño programa de ejemplo escrito en PostScript. Intenta descifrar que hace. Para comprobarlo, cópialo a un archivo y pasa ese archivo como argumento al programa `gs`, disponible en todas las distribuciones Linux:

```
/centimetro 28.3447 def
/angulo 4 def
/radio 10 def
/gris 0 def
10.5 centimetro mul 14.9 centimetro mul translate
4 4 scale
newpath
/Times-Roman findfont 12 scalefont setfont
0 1 360{
        angulo rotate
        radio 0 moveto
```

```
        (X) show
        /radio radio 0.1 add def
        /gris gris 1 360 div add def
        gris setgray
} for
showpage
```

Capítulo 8

Unidades de disco (I)

8.1. Notas históricas

Gran parte de nuestra información se encuentra registrada sobre medios magnéticos: discos, cintas de audio y video, tarjetas de crédito, etc.

Las memorias magnéticas son el soporte por excelencia, desde hace más de tres décadas, para el manejo de la información técnica, científica, médica y administrativa en nuestra sociedad. Sobre otros medios de registro tiene la ventaja de que pueden reescribirse una y otra vez.

Aunque descubierta hace más de un siglo, la grabación magnética vivió en un segundo plano durante buena parte del siglo XX, debido a dificultades técnicas y ausencia de aplicaciones prácticas. Fué el descubrimiento de las relaciones entre electricidad y magnetismo el que abrió la puerta a aplicaciones como el telégrafo y el teléfono, aun cuando diversos fenómenos eléctricos y magnéticos eran conocidos aisladamente desde la antiguedad.

En 1898, el ingeniero danés Valdemar Poulsen grabó la voz humana sobre un material magnético. Con una visión de futuro que ahora podemos apreciar, Poulsen pensó en la utilidad de grabar mensajes telefónicos. En esencia, el aparato de Poulsen consistía en un cable de acero tenso, y un micrófono telefónico conectado a un electroimán, el cual deslizaba sobre el cable mientras hablaba ante el micrófono. El micrófono transformaba la voz en una corriente eléctrica variable y el electroimán a esta en un campo magnético cuya variación dependía del volumen y el tono de la voz y que quedaba registrado sobre el cable de acero. Sustituyendo el micrófono por un auricular

y deslizando de nuevo el electroimán, conseguía que el aparato funcionase a la inversa, reproduciendo los sonidos grabados.

Oposiciones políticas e industriales impidieron a Poulsen patentar su *telegráfono*, que sin embargo tenía indudables ventajas técnicas sobre el fonógrafo, inventado veinte años antes. En particular, el sonido era claro, libre de la mayor parte del ruido que sufría la reproducción fonográfica.

Por temor a la pérdida de confidencialidad que podía propiciar la grabación de conversaciones, el temor al impacto que su comercialización podía tener sobre el teléfono y la sospecha de que se usó en actividades de inteligencia militar durante la primera guerra mundial, se frenó su desarrollo en los Estados Unidos.

En cuanto a Europa, el alemán Kurt Stille mejoró el telegráfono aumentando la duración de las grabaciones y usando amplificadores que solucionaron el problema del bajo volumen de reproducción. Algunas compañías trabajaron con licencia de Stille, como la Ludwig Blattner Picture Company de Gran Bretaña, fabricante del Blattnerphone, que se usó en emisoras de radio hasta 1945.

La empresa Lorenz, también con licencia Stille comercializó una grabadora sobre cinta de acero que permitió la transmisión de programas de radio previamente grabados.

Estas primeras grabadoras precisaban de larguísimas cintas de acero, de kilómetros de longitud. El siguiente paso lo dió Kurt Pfleumer, al desarrollar un procedimiento para recubrir una cinta de papel con partículas ferromagnéticas. En 1928 fabricó una grabadora que las usaba. Las patentes de Pfleumer fueron adquiridas por la compañia AEG. Esta empresa, en colaboración con I. G. Farben, actualmente BASF, investigó con distintas partículas y sustratos, encontrando una óptima combinación de óxido de hierro sobre un sustrato plástico. Estas investigaciones permtieron la comercialización del magnetofón.

Tras el final de la segunda guerra mundial se produjo un trasvase tecnológico entre Europa y los Estados Unidos que permitió recuperar a estos últimos el terreno perdido mediante la investigación de firmas como Ampex y 3M.

La grabación magnética pronto se impuso en el cine y la radio, pero no fueron estos los únicos terrenos en que se extendió. Otro de los frutos de la investigación militar fueron los primeros ordenadores, con memorias de núcleos de ferrita, como el Whirlwind I, mientras que otros, como el UNIVAC I usaron cintas magnéticas. Otros modelos empleaban tambores y discos magnéticos. Aunque los circuitos integrados sustituyeron a las memorias de núcleos, los discos duros siguieron siendo el principal soporte de información en los ordenadores.

8.2. La crisis de las memorias masivas

El perfeccionamiento de los discos duros puede considerarse el paradigma de cómo la tecnología punta puede hacerse accesible al conjunto de la población cuando se produce en grandes cantidades. Desde los años 80, las capacidades de los discos duros han aumentado desde unos pocos Megabytes hasta centenares de Gigabytes: un crecimiento de cuatro órdenes de magnitud, al tiempo que el precio por Gigabyte se ha reducido en otros tres órdenes de magnitud.

Los discos duros tienen su origen en la unidad RAMAC de IBM, presentada en 1956. Esta unidad usaba cincuenta platos coaxiales de aluminio de unos sesenta centímetros de diámetro, recubiertos de óxido de hierro. El RAMAC tenía una capacidad de cinco Megabytes, pesaba casi una tonelada y su volumen era del orden de un metro cúbico.

Desde entonces, las mejoras en capacidad y velocidad de lectura y escritura han tenido el factor común de la miniaturización, especialmente en lo que concierne a los cabezales.

Los primeros cabezales eran de ferrita, pero en 1979 hicieron su aparición cabezales de película delgada, que lograban inscribir datos en dominios magnéticos más pequeños. El siguiente paso fueron los cabezales de efecto magnetorresistivo. Este fenómeno, ya observado por Lord Kelvin en 1857, consiste en la alteración de la resistencia eléctrica de un material cuando se expone a un campo magnético. Así, si los cabezales de ferrita usaban el fenómeno de la inducción tanto para la lectura como para la escritura, desde la introducción de los cabezales magnetorresistivos en 1995 se usa este efecto para la lectura. En 1998 se descubrió el efecto magnetorresistivo gigante en materiales estratificados. En estos materiales, la variación en la resistencia

eléctrica es uno o dos órdenes de magnitud mayor que la variación que experimentan los materiales magnetorresistivos de la generación anterior. Desde entonces, se ha descubierto toda una familia de efectos magnetorresistivos.

Sin embargo, estos avances en el diseño de los cabezales carecería de utilidad si el sustrato magnético no fuese capaz de almacenar la densidad que el cabezal puede escribir. En la búsqueda de materiales capaces de soportar densidades mayores de datos, se intenta reducir el tamaño de los dominios magnéticos. Hasta ahora, un bit puede ser almacenado en dominios que contienen del orden de 10^3 cristales de material ferromagnético, y se investiga la forma de reducir esta cantidad.

Sin embargo, existe un límite físico: cuando la energía magnética almacenada en un dominio sea del orden de su energía térmica, la magnetización no será estable. A este efecto se le llama *efecto super-paramagnético*, y se cree que se manifiesta con densidades de alrededor de veinte Gigabytes por centímetro cuadrado. Durante el presente año, Hitachi e IBM han presentado alternativas que permiten superar este límite. En concreto, IBM usa para el sustrato un material compuesto por tres capas. Dos de material ferromagnético y una intermedia, extraordinariamente delgada, de Rutenio. Por su parte, Hitachi ha presentado una técnica de grabación perpendicular mediante la cual los dominios magnéticos se incriben en el disco perpendicularmente a su superficie. Esta técnica implica el uso de cabezales especiales.

Pero la densidad máxima que puede almacenarse depende también del número de pistas por centímetro que puedan grabarse, que a su vez es función de la capacidad de resolución del cabezal y de su mecanismo de posicionamiento. Para dar una idea de la importancia de este último factor baste decir que la diferencia esencial en cuanto a capacidad entre un disco flexible de 1.44 Megabytes y una unidad LS-120, aparecida a finales de los 90 con una capacidad de 120 Megabytes, consiste en que esta última usaba un mecanismo de posicionamiento óptico.

Otro factor digno de considerar es el de la velocidad de acceso a los datos. Históricamente, el aumento de capacidad de los discos duros ha sido mucho mayor que el aumento en la velocidad de acceso, aunque la diferencia se ha disimulado con la adopción de *cachés* integradas en los propios discos, las *cachés* del sistema operativo, posibles a su vez por el abaratamiento de la memoria principal, y la optimización en los sistemas de archivos. Pero se apunta ya a la posibilidad de que se produzca una bifurcación en el mercado

entre unidades de alta capacidad y unidades de alta velocidad de acceso. Estas últimas se caracterizarán por una velocidad de rotación superior a las 10.000 rpm, lo que implicará a su vez el uso de cojinetes fluidos para los ejes.

8.3. Fundamentos físicos

8.3.1. Ley de Biot-Savart

La ley de Biot-Savart permite calcular el campo magnético creado por una corriente eléctrica. Más exactamente, establece el campo magnético creado por un elemento de corriente en un punto cualquiera del espacio. Cuando se conoce la distribución de corriente, es posible integrar para todos sus elementos, obteniendo el campo total en el punto de interés. La forma de la ley de Biot-Savart es la siguiente:

$$d\bar{\mathbf{B}} = k_m \frac{I\bar{\mathbf{dl}} \times \mathbf{r}}{r^2} \tag{8.1}$$

Donde I es la intensidad que circula por el conductor, $\bar{\mathbf{dl}}$ un elemento de dicho conductor, $\mathbf{r}$ el versor que apunta desde el elemento de corriente al punto del espacio en donde quiere calcularse el campo magnético y r la distancia entre el elemento de corriente y dicho punto. Por integración, puede entonces encontrarse el campo magnético total. Como ejemplo, consideremos el caso de un conductor rectilíneo de longitud infinita que coincide con el eje x. Calculemos el campo magnético en un punto situado sobre el eje y, a una distancia a del origen. Por simetría, este campo será el doble del generado por la porción positiva del eje x. Para un elemento genérico de este eje, situado a una distancia l del origen, el campo generado en el punto de interés viene dado por

$$d\bar{\mathbf{B}} = k_m \frac{\bar{\mathbf{dl}} \times \mathbf{r}}{r^2} \tag{8.2}$$

En nuestro caso

$$\bar{\mathbf{dl}} = \begin{bmatrix} dl \\ 0 \\ 0 \end{bmatrix} \tag{8.3}$$

$$\mathbf{r} = \frac{1}{l^2 + a^2} \begin{bmatrix} -l \\ a \\ 0 \end{bmatrix} \tag{8.4}$$

Efectuando el producto vectorial e integrando desde 0 a ∞ encontramos que el campo magnético total tiene sólo componente B_z y viene dado por

$$B_z = -2k_m \int_0^\infty \frac{ldl}{(l^2+a^2)^{\frac{3}{2}}} = \frac{2k_m}{a} \tag{8.5}$$

Vemos que el campo decrece lentamente, con la inversa de la distancia al eje. Esta ley de decrecimiento del campo con la distancia depende de la geometría del problema, como se verá en este segundo ejemplo, donde calcularemos el campo producido por una espira. En efecto, sea una espira circular de radio R contenida en el plano $x-y$ y con su centro coincidente con el centro del sistema de referencia. Calcularemos el campo magnético sobre un punto del eje z situado a una distancia a del plano $x-y$. Supondremos que la corriente circula en sentido contrario a las agujas del reloj. El vector $\mathbf{\bar{dl}}$ de un elemento de corriente en un punto de la espira tal que su radio vector forma un ángulo θ con el eje x viene dado por

$$\mathbf{\bar{dl}} = R \begin{bmatrix} -\operatorname{sen}\theta \\ \cos\theta \\ 0 \end{bmatrix} d\theta \tag{8.6}$$

y el versor $\mathbf{r}$ viene dado por

$$\mathbf{r} = \frac{1}{R^2+z^2} \begin{bmatrix} -R\cos\theta \\ -R\operatorname{sen}\theta \\ 0 \end{bmatrix} \tag{8.7}$$

Es fácil comprobar que el campo neto en las direcciones x e y es nulo, y que la componente z vale

$$B_z = \frac{2\pi k_m I R^2}{(R^2+z^2)^{\frac{3}{2}}} \tag{8.8}$$

8.3.2. Tipos de materiales

Por su comportamiento cuando son expuestas a un campo magnético externo, las sustancias pueden dividirse en 'diamagnéticas', 'paramagnéticas' y 'ferromagnéticas'.

Las primeras tienen la propiedad de atenuar ligeramente en su interior el campo externo. Materiales diamagnéticos son gases raros y sales de metales. Además, el comportamiento diamagnético es totalmente independiente de la temperatura.

En los materiales paramagnéticos el comportamiento es similar. Cuando se introduce una sustancia paramagnética, como aluminio u oxigeno líquido, en un campo magnético, la sustancia intensifica en su interior el campo externo, pero de nuevo débilmente (0.05 % máximo). Además, cuanto más baja es la temperatura, más acusado es el efecto.

Diamagnetismo y paramagnetismo tienen la característica común de que el grado de magnetización depende de la existencia de un campo externo. Si este campo externo desaparece, también lo hará la magnetización de la sustancia que hayamos colocado en su interior. Por consiguiente, tanto los materiales diamagnéticos como los paramagnéticos son inadecuados para el almacenamiento a largo plazo de la información.

Las sustancias ferromagnéticas son de mayor importancia tecnológica. En estos materiales, pequeñas áreas llamadas *dominios*, que contienen miles de millones de átomos, se encuentran totalmente magnetizadas. Sin embargo, a escala macroscópica este efecto pasa desapercibido, pues unos dominios se compensan con otros de magnetización inversa. Pero si una sustancia ferromagnética se introduce en un campo magnético, todos los dominios se alinean con el mismo, y el resultado es una notable amplificación del campo magnético interno. Para los propósitos de la grabación de datos, el efecto que nos interesa es el de la *remanencia*, en virtud del cual la sustancia permanece magnetizada aún después de que el campo externo se haya desconectado.

Se ha encontrado además que la magnetización del material es proporcional a la intensidad del campo externo, pero que llegados a cierto punto una intensificación del campo externo no lleva aparejada la correspondiente intensificación del campo en el interior del material. Se dice entonces que se ha alcanzado la saturación. Sin embargo, cuando el campo externo se va haciendo más pequeño, el campo interno también se hace más pequeño, pero la respuesta es proporcionalmente menor que cuando el campo externo estaba aumentando. Como resultado, aunque el campo externo se haga disminuir hasta cero, la magnetización del material no se hará cero, sino que quedará una *magnetización remanente*.

Para conseguir anular la magnetización remanente, es preciso aplicar un campo magnético de sentido contrario, llamado *campo coercitivo*.

Al ciclo descrito se le llama *histéresis*, y se sigue de él que para escribir datos sobre una superficie ferromagnética, mediante la magnetización de una porción de ella en una dirección dada, es preciso aplicar un campo ligeramente mayor, como mínimo, al campo coercitivo.

Por otra parte, dominios adyaccentes en direcciones opuestas pueden influirse mútuamente, llegando a desmagnetizarse en un caso extremo. Por eso, elevadas densidades de almacenamiento necesitan sustancias con una histéresis más acusada, es decir, con un campo coercitivo mayor, lo que a su vez implica CLE capaces de generar campos más intensos.

Otra característica de las sustancias ferromagnéticas es que existe una tempertura bien definida, llamada *punto de Curie* por encima de la cual el comportamiento ferromagnético desaparece brúscamente, y el material pasa a comportarse como paramagnético. Esta tabla recoge los puntos de Curie para algunas sustancias:

Sustancia	Temperatura de Curie
Hierro	770
Cobalto	1121
Níquel	358
Gadolinio	20
Fe(65)Co(35)	920
MnBi	360 (Aleación)

Falta por hablar sobre el fenómeno de la inducción, que es el que permite escribir y leer sobre superficies ferromagnéticas. Consiste en esencia en que un campo magnético variable induce una corriente eléctrica. Por ejemplo, si colocamos un anillo de material conductor en un campo magnético variable, se producirá en el anillo una corriente eléctrica, también variable. A la inversa, una corriente eléctrica constante crea un campo magnético constante, y una corriente variable crea un campo variable.

De esta manera, si exponemos un conductor cerca de la superficie de un disco en rotación, el campo magnético variable creará en el conductor una corriente variable, que puede interpretarse como una corriente de bits. A la inversa, si colocamos el conductor cerca de la superficie y hacemos que circule

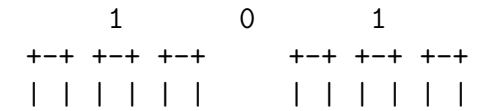
una corriente eléctrica a través de él, se creará un campo magnético, que si es superior al campo coercitivo magnetizará en una determinada dirección la zona más próxima.

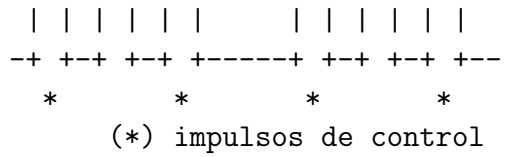
Hasta 1995, se usaron cabezales de lectura/escritura (CLE en adelante) basados exclusivamente en la inducción magnética, pero en esta fecha aparecieron los cabezales magneto-resistivos. En uno de estos cabezales la lectura se realiza midiendo cambios microscópicos en la resistencia de un material del que está construido el cabezal cuando es expuesto a un campo magnético. En 1998 se produjo otra mejora importante en la construcción de este tipo de cabezales, produciéndose cabezales magnetorresistivos con una sensibilidad muy superior, que dieron en llamarse cabezales magnetorresistivos gigantes.

Se suele decir que las informaciónes en el disco se guardan como una secuencia de estados de magnetización y no magnetización. Esto no es exacto, ya que las partículas magnéticas son muy pequeñas, y de tamaño no uniforme, de forma que ¿ cómo podría saber el cabezal si acaba de leer tres o cuatro unos seguidos ?.

En lugar de esto, habría que asignar estados de magnetización a zonas finitas, y el tamaño de una zona se le indicaría al cabezal mediante un pulso de reloj, de manera que cada nuevo *tic* indicase la entrada de una nueva zona. Sin embargo, la velocidad del disco duro no es perfectamente constante, y por otro lado no se puede magnetizar una número exacto de partículas, sino que este número es variable.

Sin embargo, si que pueden medirse cambios de magnetización. El paso de una zona magnetizada a otra no magnetizada se puede interpretar como un 1, y la ausencia de este cambio como un cero. Pero esto reproduce el problema anterior, ya que ¿ cómo sabe el cabezal cuantos cambios de flujo podían haberse producido y no lo hicieron, indicando con ello cuantos ceros consecutivos se han leido ?. Nuevamente, sería necesario un pulso de reloj. Para solucionar el problema, se graba directamente la secuencia de *tics* de reloj en el disco duro, y los datos se incluyen en esta secuencia: un cambio de flujo para un uno y ningún cambio para un cero. A este procedimiento se le llama FM (Frecuency Modulation), y se explica en la figura:





Obsérvese que existen más pulsos de control que de datos, lo que limita la capacidad de almacenamiento por unidad de superficie. Por este motivo se desarrollaron esquemas alternativos de codificación, como el MFM y RLL, que aumentan la densidad de información recurriendo a codificaciones alternativas del flujo de bits y los impulsos de control junto con electrónicas mejoradas que aseguran velocidad de rotación casi constante.

8.4. Geometría y parámetros dinámicos del disco

Comenzaremos la discusión admitiendo para las unidades de disco la geometríia que se les asignó originalmente. Esto nos permitirá fijar ideas e introducir conceptos, y más adelante veremos la forma en que esta geometría básica ha cambiado con el tiempo, y las razones para ello.

En esencia, un disco consiste en un conjunto de platos circulares, paralelos entre sí, solidarios a un eje que pasa por el centro de todos ellos. El eje se pone en rotación y un CLE se desplazada sobre cada una de las superficies disponibles, escribiendo y leyendo datos. Cuando un CLE se encuentra quieto, puede leer y escribir datos sobre una pista circular. Al conjunto de todas las pistas, sobre todas las caras de todos los platos, del mismo radio es a lo que se llama un *cilindro*. En cuanto a las pistas individuales, se encuentran divididas en sectores que tradicionalmente han sido de 512 bytes. Sin embargo, el disco ha de albergar mucha más información adicional. Así, cada sector ha de llevar asociado su dirección en el disco, códigos de redundancia para detectar y recuperar errores en este dirección, los propios datos, códigos de redundancia para detectar y recuperar errores en los datos y saltos entre sectores para dar tiempo al cabezal a encenderse y apagarse. Sin estos saltos, el ruido en el CLE podría entorpecer la lectura de la dirección del sector siguiente. En total, todos estos datos adicionales y espacios en blanco consumen alrededor de 20 % del espacio en disco. Esta es como decimos la geometría tradicional de los discos, y con esta geometría un sector se localiza por el cilindro al que

pertenece, por la pista dentro de ese cilindro y por el sector dentro de esa pista.

En un disco duro con cabezas móviles, la lectura de un sector requiere en primer lugar el desplazamiento del CLE hasta el cilindro que contenga al sector. Entonces, es preciso esperar hasta que el sector buscado pase bajo el CLE. Cuando esto sucede, ya pueden transferirse los datos, típicamente a velocidades de 320 MB/s. Resumiendo, existen tres parámetros principales en el funcionamiento del disco:

- Tiempo de busqueda, que es el tiempo necesario para colocar el cabezal sobre el cilindro adecuado.
- Tiempo de latencia rotacional, que es el tiempo preciso para que el sector buscado pase bajo el CLE. Varía desde 0 al periodo de rotación, por lo que en media es el tiempo preciso para que se complete media revolución.
- Tiempo de transferencia, que es el tiempo necesario para transferir los datos a memoria.

Llamaremos a estos tres tiempos T_B, T_L y T_T. Se cumple en general que $T_B > T_L > T_T$. Por tanto, es rentable transferir a memoria bloques más grandes, ahorrando así intervalos de búsqueda y latencia. Por el mismo motivo, tiene interés optimizar el movimiento del CLE, de manera que los desplazamientos sean mínimos. Más adelante en este capítulo expondremos algunos de los algoritmos usados con este fin. La relación entre los tres tiempos básicos citados también aconseja guardar los datos en bloques de entre algunos centenares y algunos miles de octetos, llamados *bloques*. Más adelante volveremos sobre este punto.

Consideremos la lectura de n bloques secuenciales (consecutivos) sobre una unidad de disco. El tiempo invertido es:

$$t_s = T_B + T_L + nT_T \tag{8.9}$$

Sin embargo, si los bloques se encuentran distribuidos aleatoriamente sobre el disco, el tiempo invertido en su lectura será:

$$t_a = n(T_B + T_L + T_T) \tag{8.10}$$

y vemos que, cuando n se hace muy grande, la relación entre ambos tiempos tiende a:

$$\frac{t_s}{t_a} = \frac{T_T}{T_B + T_L + T_T} \tag{8.11}$$

Consideremos ahora que la unidad mínima de información que leemos no es un sector, sino un registro de un archivo. Si B es el número de octetos por sector y R el número de octetos por registro, llamemos $b = B/R$, que representa el número de registros contenidos en cada sector. Si el archivo contiene k registros, el número de sectores ocupados por dicho archivo es $kR/B = k/b$.

Como se intuye fácilmente, el que la unidad mínima de lectura coincida o no con el tamaño del sector no tiene influencia cuando se trata de lecturas secuenciales, pero sí la tiene cuando se trata de lecturas aleatorias. Veámoslo con un ejemplo extremo en que la unidad mínima de lectura/escritura es tan grande que contiene muchos registros. Llamemos *grupo* a esta unidad. Cuando se proceda a leer un gran cantidad de registros de forma aleatoria será preciso visitar varias veces el mismo grupo, con el consiguiente gasto en tiempo de búsqueda. Pero cuando la lectura es secuencial, el tamaño del grupo carece de importancia, puesto que el inicio del mismo se busca una sola vez.

Al contrario, cuando la lectura se realiza no por registros sino por grupos, cuanto mayor sea el tamaño del grupo más eficiente es la operación, puesto que cada grupo contiene un mayor número de registros.

El problema es que normalmente se efectúan lecturas y escrituras tanto por grupos como por registros. Supongamos que de P registros se hacen αP actualizaciones manuales de registros individuales, que consumen un tiempo $\alpha P(T_B + T_L + T_T)$, donde ahora hay que entender que T_T es el tiempo de transferencia de un grupo, como unidad mínima de lectura/escritura. Supongamos además que cada cierto tiempo se lee el archivo entero. Si llamamos Q al número de octetos por grupo y S al número de octetos por registro, entonces la lectura completa del archivo por grupos, más la lectura aleatoria de una fracción α de los registros, lleva un tiempo:

$$\left(\frac{PS}{Q} + \alpha P\right)(T_B + T_L + T_T) \tag{8.12}$$

Ahora bien, T_T es proporcional a Q. Si llamamos β a dicha constante de proporcionalidad:

$$\left(\frac{PS}{Q} + \alpha P\right)(T_B + T_L + \beta Q) \tag{8.13}$$

Esta función de Q tiene un mínimo cuando

$$Q = \sqrt{\frac{S(T_B + T_L)}{\alpha\beta}} \tag{8.14}$$

8.5. El teorema de Salzberg

Demostraremos que el tiempo medio de búsqueda es igual al tiempo necesario para atravesar un tercio de los cilindros del disco. Llamemos $s(i)$ al tiempo necesario para atravesar i cilindros. Este tiempo es aproximadamente igual al tiempo necesario para levantar el CLE más i veces el tiempo k necesario para cambiar de un cilindro al siguiente:

$$s(i) = c + ki \tag{8.15}$$

Llamaremos $p(i)$ a la probabilidad de que el CLE se desplaze i cilindros. Entonces, si el disco tiene N cilindros:

$$s = \sum_{i=0}^{i=N-1} s(i)p(i) \tag{8.16}$$

Con N cilindros, tenemos N orígenes y N destinos posibles, luego N^2 posibles movimientos, de los cuales N consisten en no moverse de cilindro, luego:

$$p(0) = \frac{N}{N^2} = \frac{1}{N} \tag{8.17}$$

pero en este caso $s(0) = 0$, luego podemos prescindir del primer término de la sumatoria. Recordemos que $p(i)$ es el número de formas en que podemos desplazarnos i cilindros dividido entre el número total de formas en que podemos desplazarnos. En general, para i entre 1 y N existen $N - i$ formas de desplazarse hacia el centro del disco, y otras tantas de desplazarse hacia afuera, luego:

$$p(i) = 2\frac{N-i}{N^2} \tag{8.18}$$

de donde

$$s = 2\sum_{i=1}^{N-1} \frac{(c+ki)(N-i)}{N^2} \tag{8.19}$$

Desarrollando esta expresión y teniendo en cuenta las fórmulas para la suma de enteros consecutivos y de la suma de los cuadrados de enteros consecutivos:

$$\sum_{i=1}^{i=N} i = \frac{N(N+1)}{2} \tag{8.20}$$

$$\sum_{i=i}^{i=N} i^2 = \frac{N(N+1)(2N+1)}{6} \tag{8.21}$$

llegamos a la expresión:

$$s = c - \frac{c}{N} + \frac{kN}{3} - \frac{k}{3N} \tag{8.22}$$

Cuando N es grande, lo que es el caso normal,

$$s = c + k\frac{N}{3} \tag{8.23}$$

que es lo que deseábamos demostrar.

8.6. Cambios en la geometría y en la interfaz

El objetivo de un disco es almacenar información de forma persistente, y servirla a petición del sistema operativo. Los diseñadores de las unidades de disco se preocupan de que este proceso se realize de forma fiable y rápida, y de que la cantidad de información que puede almacenarse por unidad de superficie sea lo más grande posible, y todo ello con el coste mínimo.

Geométricamente, un disco es un espacio tridimensional donde se organiza el almacenaje de los datos. Cada superficie de un plato es un espacio bidimensional, y el apilamiento de un conjunto de estos platos proporciona una tercera dimensión. Una forma de incrementar la capacidad de los discos es simplemente aumentando la cantidad de platos, pero esto hace al sistema de disco más vulnerable a las vibraciones, aumenta el costo y produce otros efectos indeseados, como el flujo turbulento de aire en el interior del dispositivo, lo que tiene un efecto negativo sobre el mecanismo de control del CLE. Así, a finales de los 90 los discos de gama alta tenían en torno a los diez cabezales, y los discos para el mercado de consumo (ordenadores personales) tenían en torno a cuatro. Alrededor de 2003 los discos de gama alta ya sólo tenían cuatro o cinco cabezales, y los de gama baja uno o dos. Por ese motivo se ha preferido aumentar la densidad de pistas, haciéndolas más próximas las unas a las otras, es decir, aumentando el número de pistas por centrimetro a lo largo del radio del disco y también aumentando el número de sectores por pista, es decir, aumentando la densidad de datos en cada pista. El producto del número de pistas por centímetro y del número de sectores por pista proporciona la densidad de datos por unidad de superficie. A principios de los 80, esta densidad era del orden de los 1.5 MB/cm^2. A principios de los 90 era de 10 MB/cm^2, y en los primeros años de este siglo ha crecido hasta los 1000 MB/cm^2.

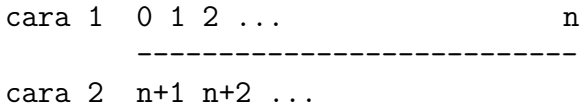
Durante un tiempo, los fabricantes de discos se adaptaban a la geometría que hemos explicado antes: los discos constan de pistas que se dividen en sectores. Todas las pistas tienen el mismo número de sectores y las pistas de distintas caras que tiene el mismo radio se agrupan en cilindros. Así el trío (Cabeza,Cilindro,Sector) especifica totalmente el lugar donde un sector se localiza. La geometría estaba expuesta al sistema operativo, y era éste quien había de lidiar con sus peculiaridades. Por ejemplo un fabricante podía tener un disco con una determinada capacidad. Otro fabricante podía tener un disco de capacidad similar, pero con menos densidad de pistas y a cambio mayor densidad de sectores por cada pista. Un tercero podía tener baja densidad de lo uno como de lo otro, pero compensarlo añadiendo un plato adicional. Además, los sectores defectuosos eran un grave problema. Cuando la densidad por unidad de superficie era baja, a finales de los 70, era posible fabricar discos perfectos, sin sectores defectuosos. Pero a medida que aumentó la densidad de datos, esto resultó imposible, y los sistemas operativos tuvieron que habérselas con la forma de gestionar los sectores defectuosos. La forma de hacerlo fue manteniendo una lista de sectores defectuosos, y haciéndoles corresponder sectores sanos. De esta forma cuando el sistema

recibía una petición para un sector, no iba directamente a ese sector, sino que usaba el número de sector para indexar una tabla que indicaba el sector sobre el que efectivamente se realizaría la lectura o escritura.

Pero entonces surgió otro problema: los fabricante comenzaron a colocar más sectores en el exterior del disco, y menos en el interior. Era lógico. Una pista cercana al borde del disco tiene una longitud del orden del 50 % mayor que una pista cerca del centro, y por tanto no tenía sentido mantener el mismo número de sectores por pista. Entonces, el disco se divide en zonas y las pistas de cada zona contienen el mismo número de sectores. Las zonas exteriors, más; las zonas interiores, menos. Un disco moderno puede tener entre 15 y 25 zonas.

Esto obligó a ocultar al sistema operativo la geometría de cada disco: era demasiado compleja, y había demasiados discos distintos. Pero para poder olvidarse de la geometría real del disco hubo que cambiar la interfaz de programación. En los viejos tiempos, el programador pasaba a la controladora las coordenadas de un sector. En los nuevos tiempos, el disco se ve como una sucesión lineal de sectores, como si fuese una cinta magnética y sólo hay que decir qué sector quiere leerse. Este es el fundamento de las interfaces ATA y SCSI.

De esta forma, la importancia del cilindro ha desaparecido prácticamente. Importaba hace un par de décadas diseñar sistemas de archivos que colocasen los datos de tal forma que los archivos ocupasen preferentemente un mismo cilindro, evitando así el movimiento de los cabezales. Actualmente, simplemente el programador ignora donde están los cilindros, y en muchos casos ni existen. Por ejemplo, alrededor del 40 % de los discos que se venden hoy en día llevan un único CLE, un único plato con datos por una cara y esa cara dividida en un par de docenas de zonas con números de sectores por pista distintos ¿qué sentido tiene hablar de cilindros ? Hay otros cambios que también contribuyen a diluir el concepto de cilindro. El más evidente consiste en numerar alternativamente las pistas. En la numeración convencional las pistas se numeran desde el exterior al interior, y cuando se acaba una cara se comienza con la siguiente, de nuevo numerando desde el interior al exterior, como muestra la figura:



Resulta sorprendente que se tardase tanto tiempo en numerar las pistas de la segunda cara desde el interior hacia el exterior. De esta forma el movimiento del cabezal es mínimo, y una lectura que incluya a la pista n puede hacerse secuencialmente, sin necesidad de desplazar el cabezal al comienzo del disco en mitad de la operación.

```
cara 1  0 1 2 ...                  n
        --------------------------
cara 2                 ... n+3 n+2 n+1
```

Este cambio en la geometría que arrastra consigo el cambio en el modelo de programación ha sido posible gracias al aumento de la densidad de datos por unidad de superficie. Otro de los efectos de este incremento en la densidad ha sido la reducción en el tamaño de los discos. La razón es energética. Un disco de radio R y masa m tiene un momento de inercia

$$I = \frac{1}{2}mR^2 \tag{8.24}$$

y cuando gira con velocidad angular ω su energía cinética es

$$T = \frac{1}{2}I\omega^2 = \frac{1}{4}mR^2\omega^2 \tag{8.25}$$

Cuando un disco parte del reposo y alcanza su velocidad de funcionamiento en un tiempo Δt, la potencia consumida es

$$P = \frac{T}{\Delta t} \tag{8.26}$$

y como se ve crece con el cuadrado del radio. Un disco con un radio la mitad que otro consume, para alcanzar la misma velocidad, cuatro veces menos energía. A la inversa, para un consumo dado de energía, un disco más pequeño puede girar mucho más rápido. Por ejemplo, un disco que gira a 5400 rpm tiene un diámetro de 95 mm y un tiempo de pista a pista de 9 ms. Un disco que gira a 15.000 rpm tiene un diámetro de 65-70 mm y un tiempo de pista a pista de 3 ms. La reducción de tamaño no sólo favorece mayor velocidad de giro, sino menor tiempo de pista a pista.

Un problema adicional vino con los cabezales magneto-resistivos. En uno de estos cabezales, existe un elemento magneto-resistivo para la lectura, pero sigue habiendo un elemento inductivo para la escritura. Están muy cerca el uno del otro, pero no pueden obviamente estar simultáneamente en el mismo sitio. Como la densidad de pistas por centímetro era ya muy alta, era preciso poner en el mismo disco información para el control del servo. Es decir, en lugar de control en lazo abierto (el servo recibe una orden y la ejecuta "a ciegas") se pasó a control en lazo cerrado: sobre el disco parte de la información sirve para guiar el movimiento del cabezal hacia el sitio correcto. Pero con una densidad de pistas alta, la información para el servo cuando se realiza una escritura no es la misma que cuando se efectúa una lectura, porque han de hacerse con elementos distintos que están en posiciones distintas. Esto era un paso en el sentido opuesto al que se pretendía: al aumentar la densidad de pistas era preciso añadir mucha información para que el servo del CLE pudiese colocarse en la posición correcta, pero entonces, buena parte de la ganancia en capacidad por el aumento de la densidad de pistas se perdía dedicándola al control.

8.7. Fiabilidad y prestaciones

Las interfaces de más alto nivel, al dejar en manos del propio disco muchas operaciones, se benefician de la habilidad de éste para hacer cosas que el propio sistema no puede hacer. Por ejemplo, consideremos la gestión de errores. Un sector lleva asociados unos bytes de detección y corrección de errores. Cuando el disco detecta un error en la lectura de un sector, puede hacer (en un disco moderno) entre veinte y treinta intentos de lectura: moviendo el CLE ligeramente a la izquierda, o ligeramente a la derecha, o comenzando a leer una fracción de milisegundo antes o después, y combinando ambas técnicas. Este es un procedimiento que se efectúa muy rara vez, pero es preciso implementarlo, porque es la diferencia entre poder recuperar los datos y no poder hacerlo. Una vez que el disco ha conseguido recuperar los datos, puede moverlos a una sección de disco sana, marcando como defectuosa la original. Una cierta porción de la capacidad total de un disco moderno se dedica a este menester.

Otra de las ventajas de disponer de una interfaz de alto nivel y un control sofisticado en el propio disco es la posibilidad de que el disco almacene en un *buffer* los datos hasta que el sistema pueda leerlos, o de que almacene peticiones de lectura mientras está escribiendo. De esta forma, se elimina la necesidad de sincronización entre el sistema operativo y el disco.

En entornos sometidos a una carga alta de peticiones, la posibilidad de que el disco almacene una cola de peticiones y pueda analizar esta cola y servirla de la mejor forma posible incrementa notablemente el rendimiento. Así, el disco, con su conocimiento de la geometría real puede alterar el orden en que recibió las peticiones para maximizar el número de transferencias por segundo. Las posibilidades de optimización son tanto mayores cuanto mayor es la longitud de la cola. Por ejemplo, con una cola de unas pocas órdenes un disco moderno puede servir del orden de doscientas peticiones por segundo. Con una cola de un centenar de elementos se pueden servir del orden de 400 peticiones por segundo y con una cola de 250 elementos más de 500 peticiones. En un entorno donde existan varios procesos accediendo al disco, cada uno con su propia cola, la controladora del disco puede unir varias colas en una, incrementando su longitud y por tanto mejorando, como hemos visto, el número de peticiones por segundo que pueden servirse.

Esto tiene un impacto adicional sobre los sistemas de archivos: hace innecesarios los algoritmos de movimiento del cabezal. Supongamos que un sistema operativo tiene una cola de peticiones; cada petición especifica el sector que desea leer. El sistema de archivos ordena por número de sector, suponiendo que los sector con número consecutivos están próximos unos de otros y que por tanto vá a minimizar el tiempo de servicio. Esta puede ser una suposición totalmente falsa, ¿porqué? Porque dos sectores consecutivos en esa hipotética lista pueden estar en puntos diametralmente opuestos en la pista, y es preciso, después de leer un sector, esperar un tiempo muy grande (relativamente) hasta que el siguiente sector pasa bajo el cabezal. Pero durante ese tiempo el cabezal quizás podría desplazarse a pistas cercanas, hacer alguna lectura pendiente de la cola y volver a la pista original. Evidentemente, esto es algo que sólo puede decidir el controlador del disco *in situ*.

8.8. El tamaño del sector

El último vestigio de lo que hemos llamado *viejos tiempos* es el tamaño del sector fijo en 512 bytes. Uno de los componentes básicos de un sistema de disco es el mecanismo de recuperación de errores, que se basa en la interpretación de los códigos de detección y correción que acompañan a cada sector. De esta forma, a cada 512 bytes se solían añadir 16 bytes adicionales que hacían esta función. Pero a medida que ha aumentado la densidad superficial, al haber más bits por unidad de superficie, los errores de fabricación inevitables afectan cada vez a mayor número de bytes, y esto ha obligado, en los diez últimos años, a duplicar el número de bytes dedicados a control

de errores. Previsiblemente, la proporción de los discos dedicados a control de errores seguirá creciendo, y por eso todos los fabricantes desean tamaños de sector más grandes. Se ha propuesto un tamaño de sector de 4K. Desde el punto de vista de los fabricantes, esta modificación es trivial, pero desde el punto de vista de los fabricantes de sistemas operativos el impacto es tan grande que se investiga en estos momentos la forma de resolverlo. Y sin estar resuelto este problema, se proponen ya otras posibilidades excitantes, como ocultar totalmente el tamaño de sector al sistema operativo, lo que obligaría a cambiar la interfaz de programación de nuevo pero abriría nuevas oportunidades de optimización.

Capítulo 9

Unidades de disco (II)

9.1. Jerarquías de almacenamiento

Es un hecho que la cantidad de datos que desean almacenarse es siempre mayor o igual que el espacio de almacenamiento disponible. Normalmente, se desea poder recuperar datos en el menor tiempo posible, pero, al mismo tiempo, mantener lo más bajo posible el coste del sistema. Una solución consiste en colocar en un medio de almacenamiento rápido los datos a los que se accede con más frecuencia, y en un medio más lento, pero menos costoso, aquellos datos que con menor probabilidad necesitarán ser recuperados. La cuestión que se plantea en este apartado es la de encontrar la distribución que asegura el equilibrio óptimo entre dos necesidades contrapuestas: velocidad de acceso y coste reducido.

Imaginemos que se producen datos a un ritmo medio constante, y que es preciso mantenerlos almacenados durante un periodo de tiempo dado antes de desecharlos. De forma arbitraria, asignaremos el valor 1 a este periodo de tiempo, y de la misma forma asignaremos el valor de 1 al tiempo medio de acceso y coste del primer medio. En un primer cálculo, supondremos la existencia de dos medios distintos, estando caracterizado el segundo de ellos por un tiempo medio de acceso $t > 1$ y un coste $c < 1$. Evidentemente, la distribución óptima de los datos dependerá de la probabilidad de acceso a los datos según su antigüedad. Así, si esta probabilidad decrece rápidamente con el tiempo tiene sentido mantener en el primer medio una pequeña cantidad de datos, y pasar el resto al segundo medio. A la inversa, si esta probabilidad decrece lentamente tiene sentido mantener mayor cantidad de datos en el primer medio, puesto que será relativamente probable tener que acceder a los mismos. Un modelo plausible para la distribución de probabilidad de

acceso a los datos en función de su antigüedad es:

$$p(x) = p(0)\,[1 - x^n] \tag{9.1}$$

para cada $0 \leq x \leq 1$. Aquí, n es un entero mayor o igual que la unidad, y $p(0)$, la densidad de probabilidad para $x = 0$, que viene dada por la condición de normalización:

$$\int_0^1 p(x)dx = 1 \tag{9.2}$$

de donde se sigue:

$$p(0) = 1 + \frac{1}{n} \tag{9.3}$$

Nuestro objetivo es minimizar el tiempo medio de acceso sin penalizar el precio del sistema. Sea $\mathcal{T}$ el tiempo medio de acceso, y $\mathcal{C}$ el coste total. Asignemos arbitrariamente la unidad tanto para el tiempo de acceso como el coste del primer medio, y sean c y t el coste y tiempo medio de acceso para el segundo medio. Finalmente, sea $0 \leq z \leq 1$ el intervalo de tiempo durante el cual los datos generados suponemos que a ritmo constante son almacenados en el primer medio, antes de ser traspasados al segundo.

$$\mathcal{T} = \int_0^z p(x)dx + t\int_z^1 p(x)dx \tag{9.4}$$

Se ve enseguida que, salvo términos constantes:

$$\mathcal{T} = \frac{n+1}{n}\left[(1-t)(z - \frac{z^{n+1}}{n+1})\right] \tag{9.5}$$

Y de la misma forma:

$$\mathcal{C} = z + c(1 - z) \tag{9.6}$$

Ahora, queremos minimizar la cantidad:

$$\mathcal{J} = \mathcal{T} + \alpha\mathcal{C} \tag{9.7}$$

donde α es un factor de peso. Siguiendo el procedimiento habitual, de:

$$\frac{\partial \mathcal{J}}{\partial z} = 0 \tag{9.8}$$

encontramos:

$$z = \left(1 - \frac{n}{n+1}\frac{\alpha(c-1)}{(1-t)}\right)^{\frac{1}{n}} \tag{9.9}$$

que conduce a la condición adicional para α:

$$0 \leq \alpha \leq \frac{n+1}{n}\frac{1-t}{c-1} \tag{9.10}$$

9.1.1. Múltiples niveles

Supongamos la partición del intervalo $(0, 1)$ mediante un conjunto de valores intermedios τ_i, supuesto que $\tau_i > \tau_{i-1}$. Mantenemos las suposiciones previas respecto a la probabilidad de acceso a datos de antigüedad x. Cada uno de los intervalos de tiempo puede almacenar una cierta cantidad de información, que es proporcional al tamaño del intervalo. Sean t_i y c_i los tiempos de acceso y coste por megaocteto en el intervalo (τ_{i-1}, τ_i) [1]. La probabilidad total de acceso en este intervalo viene dada por:

$$p_i = \int_{\tau_{i-1}}^{\tau_i} p(0)\,(1 - x^n)\,dx \tag{9.11}$$

El tiempo medio de acceso viene dado por:

$$\mathcal{T} = \sum_i p_i t_i \tag{9.12}$$

y el coste total del sistema por:

$$\mathcal{C} = \sum_i \gamma c_i (\tau_i - \tau_{i-1}) \tag{9.13}$$

Donde γ es una constante de proporcionalidad que convierte de unidades de tiempo a megaoctetos, suponiendo un ritmo constante en la producción de datos.

[1] Se sobreentiende: en el medio de almacenamiento en que se almacenan los datos con antigüedad entre τ_{i-1} y τ_i

Queremos minimizar $\mathcal{J} = \mathcal{T} + \beta\mathcal{C}$, y esto se consigue eligiendo un conjunto de τ_j para el cual:

$$\frac{\partial \mathcal{J}}{\partial \tau_j} = 0 \tag{9.14}$$

β es una constante que abarca tanto γ como el factor de peso deseado para $\mathcal{C}$ frente de $\mathcal{T}$. Operando:

$$\frac{n+1}{n}(t_j - t_{j+1})(1 - \tau_j^n) + \beta(c_j - c_{j+1}) = 0 \tag{9.15}$$

Definiendo:

$$\Delta t_j = t_{j+1} - t_j \tag{9.16}$$
$$\Delta c_j = c_{j+1} - c_j \tag{9.17}$$

obtenemos finalmente:

$$\tau_j = \left(1 + \beta \frac{n}{n+1}\frac{\Delta c_j}{\Delta t_j}\right)^{\frac{1}{n}} \tag{9.18}$$

Por supuesto, para cada j debe haber un $\tau_j > \tau_{j-1}$ que conduzca a:

$$\frac{\Delta c_j}{\Delta t_j} > \frac{\Delta c_{j-1}}{\Delta t_{j-1}} \tag{9.19}$$

9.1.2. Dos ejemplos

Considérese un disco Ultra Wide SCSI con interfaz de fibra óptica como un primer medio, y un disco ATA como el segundo medio. n se obtiene de forma experimental, y suponemos que vale $n = 2$. Aproximadamente, $t = 2$ y $c = 0{,}5$, que conduce al máximo permisible para α de 3. Para estos valores, $\tau_1 = \sqrt{\frac{2}{3}}$.

Como un segundo ejemplo, considérese un disco ATA como el primer medio y una cinta DAT como el segundo. Para la cinta, aproximadamente, $c = 0{,}6$ y $t = 10$, con un valor máximo de $\alpha = 33$. Manteniendo el valor $n = 2$ se obtiene $\tau_1 \simeq 0{,}15$

9.2. Sistemas de discos redundantes

Los sistemas RAID (Redundant Array of Independent Disks) fueron propuestos en los años 80 como solución al problema planteado por la cada vez mayor brecha existente entre las velocidades de los procesadores y memorias por un lado y las velocidades de los sistemas de discos, por otra. Los sistemas RAID organizan un conjunto grande de discos independientes en un único disco de gran capacidad y altas prestaciones. Estos sistemas pueden diseñarse con dos orientaciones distintas e incompatibles entre sí, lo que obliga a sopesar las distintas opciones: a) sistemas orientados al paralelismo, para aumentar la velocidad de E/S y b) sistemas orientados a la redundancia, para aumentar la fiabilidad.

Al distribuir una operación de E/S entre un número grande de discos, se consigue aumentar la velocidad de transferencia, distribuir uniformemente la carga entre los discos y eliminar los cuellos de botella que se producen cuando un canal de E/S se satura. Pero al aumentar el número de discos aumenta también la probabilidad de que alguno de ellos falle. Por ejemplo, un disco sencillo tiene un tiempo medio entre fallos de unas 200.000 horas, que son aproximadamente 23 años. Un sistema de 100 discos tiene un tiempo medio entre fallos de 2000 horas, que son aproximadamente tres meses. Por tanto, es preciso aumentar la redundancia, bien escribiendo más de una vez los mismos datos en discos distintos, bien escribiendo códigos de detección y corrección de errores, lo que perjudica a las prestaciones. Además, es difícil mantener la consistencia entre los datos y los códigos de error cuando hay múltiples accesos en paralelo a los mismos datos, o cuando se producen caídas del sistema.

Como se ha dicho, distribuyendo los datos entre un número grande de discos se favorece el paralelismo y por tanto la velocidad. Este paralelismo tiene dos aspectos: primero, peticiones independientes pueden servirse en paralelo por discos separados; segundo, peticiones individuales que involucren varios discos pueden servirse más rápido mediante la participación coordinada de los discos.

La mayoría de los sistemas RAID pueden clasificarse atendiendo a dos parámetros: a) la granularidad con que los datos son distribuidos y b) el método empleado para calcular y distribuir entre los discos la información redundante necesaria para conseguir una alta fiabilidad.

Si la granularidad es *fina* significa que un paquete de datos se divide en muchos fragmentos pequeños y se distribuye entre todos los discos del sistema. La velocidad que se consigue entonces es muy alta, pues cada disco lee una pequeña cantidad de datos en paralelo con el resto de los discos. Pero aparecen dos desventajas a) sólo puede servirse una petición lógica de E/S cada vez y b) hay que mover los cabezales de todos los discos.

Si la granularidad es *gruesa* los fragmentos son mayores, de manera que si la cantidad total de datos que se quieren leer o escribir es pequeña estarán afectados sólo unos pocos discos y podrán dedicarse los otros por tanto a servir en paralelo otras peticiones lógicas de entrada y salida; pero si la cantidad total de datos es grande, estará afectada la totalidad de los discos del sistema y estaremos en la misma situación que en el caso en que la granularidad es *fina*, con las mismas ventajas e inconvenientes.

En cuanto a la redundancia, aparecen dos problemas: cómo calcularla y cómo distribuirla entre los discos. El método de cálculo habitual es la paridad, aunque pueden usarse métodos más elaborados. Respecto a la distribución, hay dos tendencias: o se concentra la información redundante en un número reducido de discos o se usan todos los discos. En general, se prefiere usar el mayor número de discos para distribuir la información redundante, para evitar los discos donde se almacena la paridad se conviertan en discos con una carga muy superior a la media, pues han de actualizarse para cada operación de escritura, independientemente de a cuantos discos de datos afecte esta escritura.

Con todas estas consideraciones en mente, existen siete esquemas usualmente aceptados y conocidos como *niveles RAID*:

9.2.1. RAID 0: Sistema no redundante

La velocidad en escritura es máxima, ya que no hay datos redundantes que actualizar. En consecuencia, la fiabilidad es mínima. Sin embargo, RAID 0 no tiene la mejor marca en operaciones de lectura. En sistemas RAID con redundancia esta velocidad se puede mejorar moviendo el cabezar que se encuentra más cerca de los datos [2].

[2]Se tendrán en mente las consideraciones que se hicieron en el capítulo anterior sobre la evolución de los discos en los últimos años, y en particular sobre la inaccesibilidad para el software del sistema operativo a la geometría real del disco, conocida sólo por la controladora en el propio disco

9.2.2. RAID 1: Discos espejo

Toda la información se encuentra por duplicado en disco independientes. Un sistema RAID 1 consta de n discos de datos y de otros n discos que duplican exactamente a los n primeros. Para operaciones de lectura, puede usarse aquel disco con una cola de peticiones más pequeña, o con el cabezal más cerca de los datos, o con mejores prestaciones. Si falla un disco, se tiene copia exacta de los datos en el otro.

9.2.3. RAID 2: Estilo memoria ECC

La redundancia se basa en el uso de códigos de Hamming. Un conjunto de datos se divide en varios subconjuntos que se solapan, y se calcula la paridad para cada subconjunto. Un fallo en un elemento afectará a todos los subconjuntos que lo contengan. Así, si se detecta un fallo de paridad en una serie de subconjuntos, el elemento erróneo será aquel común a todos los subconjuntos. Para un sistema de cuatro discos, son precisos tres discos redundantes, uno menos que en los sistemas RAID 1. Se puede demostrar que el número de discos redundantes necesarios es proporcional al *log* del número de discos de datos. Por tanto, este sistema es tanto más eficiente cuanto más grande.

9.2.4. RAID 3: Paridad en un único disco

RAID 2 proviene directamente de las memorias ECC. Pero al contrario que con las memorias, con los discos es muy fácil averiguar qué disco ha fallado: lo dice la controladora. Por eso basta con colocar la información de paridad en un único disco. En su forma más sencilla, un *byte* se distribuye en ocho *bits*, que se escriben cada uno en un disco distinto, más el bit de paridad que va a parar a un noveno disco.

9.2.5. RAID 4: Paridad por bloques

El concepto es similar a RAID 3, pero en lugar de escribir un *bit* en cada disco se escriben bloques de tamaño predefinido B. Si una operación de lectura requiere una cantidad de datos inferior a B, sólo es preciso leer de un disco. Como la paridad ha de ser actualizada en todas las operaciones de escritura del disco que contiene los bloques de paridad, puede fácilmente convertirse en un cuello de botella.

9.2.6. RAID 5: Paridad por bloques distribuidos

Para evitar el defecto de RAID 4, los bloques de paridad se distribuyen entre todos los discos. Esto lleva a una pequeña mejora adicional, y es que todos los discos participan en operaciones de lectura, en lugar de *todos menos uno* que intervienen en los RAID 3 y 4. Por otro lado, las prestaciones de RAID 5 dependen del método que se use para distribuir los bloques redundantes.

9.2.7. RAID 6: Códigos robustos

Los sistemas anteriores se basan en el cálculo de la paridad, que puede detectar y corregir errores simples, pero no dobles. A medida que los sistemas se hacen más grandes, aumenta la probabilidad de que se den errores dobles que no pueden ser corregidos. En RAID 6 se usan códigos de detección y corrección de errores más robustos, y por tanto es adecuado para sistemas donde la seguridad sea prioritaria.

9.3. Comparación de sistemas RAID

Existen tres métricas para medir el rendimiento de los sistemas de discos [3]: velocidad, fiabilidad y precio. Ahora bien, cada una de estas puede medirse de distintas formas. Por ejemplo, hablando de velocidad puede ser más conveniente tener en cuenta *bytes* por segundo, o peticiones por segundo, o tiempo de espera medio en cola de cada petición. Pero una vez que nos hayamos puesto de acuerdo en la forma de medir cada métrica, ¿qué hemos de comparar? ¿sistemas del mismo preciso? ¿sistemas de las mismas prestaciones? ¿sistemas de la misma fiabilidad?

En particular, los grandes sistemas de discos están orientados a un gran ancho de banda, es decir, al número de *bytes* por segundo que son capaces de mover, en lugar de a minimizar el tiempo de espera de peticiones individuales. Pero es evidente que este tiempo será más bajo cuando mayor sea el ancho de banda y cuanto más eficientemente funcionen las *cachés*, a todos los niveles. Por otro lado, las prestaciones del sistema dependen más o menos linealmente con el número de discos: un sistema de 40 discos servirá aproximadamente el doble de peticiones por unidad de tiempo que un sistema de 20 discos. Por eso es preciso normalizar la velocidad respecto al coste, refiriéndonos a los *bytes* por segundo y por euro que pueden moverse.

[3] Estas consideraciones son válidas cuando se pretenden comparar sistemas informáticos en general

Otro factor a tener en cuenta es de la diferencia entre la implementación de un sistema y su configuración y uso. Por ejemplo, un sistema RAID 5 puede configurarse para que se comporte esencialmente como un RAID 1 o como un RAID 3, por lo que estos sistemas pueden considerarse como subclases de RAID 5.

En cuanto a la fiabilidad, existen modelos que permiten calcular el tiempo medio entre fallos que comportan pérdidas irrecuperables de datos. Como pueden darse fallos o combinaciones de fallos distintas que conduzcan a una pérdida de datos, habrá de darse este tiempo medio para distintas situaciones, que resulta estar comprendido entre algunas décadas y algunos siglos. Ahora bien, estos modelos se basan en la suposición de que los discos fallan de forma independiente unos de otros. Pero piénsese en el efecto de un terremoto (durante un periodo muy largo de tiempo, la probabilidad de que se produzca uno en un lugar dado puede ser muy alta) que incrementa puntualmente la probabilidad de que fallen simultáneamente varios discos a la vez, o en el fallo del hardware que controla simultáneamente a varios discos y que puede provocar fallos correlacionados. Finalmente, existen factores de muy difícil estimación, como la solidez del software que controla a todo el sistema.

Por todos estos motivos, la comparación de sistemas RAID se hace compleja. Nosotros omitiremos aquí toda esta discusión, y remitimos al artículo fundamental sobre sistemas RAID: *RAID: High-performance, reliable secondary storage*, de Peter M. Cheng y otros.

9.4. Gestión del espacio en disco

El problema de la gestión del espacio en disco tiene dos aspectos diferentes: a) determinar si un sector está o no disponible y b) determinar a qué archivo pertenece un sector, o qué sectores componen un archivo dado. Pero antes de responder a estas dos cuestiones, hay una previa. A saber, por qué motivo los discos se organizan en bloques, en lugar de tratarlos como meras secuencias de octetos. La discusión que sigue puede en su mayor parte aplicarse a la administración de la memoria.

Hemos dicho que la velocidad de acceso secuencial a sectores es mucho mayor que la velocidad de acceso aleatoria. Podríamos entonces forzar a que todos los archivos ocupasen sectores contiguos. Esta opción se ha ensayado en algunos sistemas, sobre todo en aquellos en que el tamaño del archivo a) puede predecirse y b) permanece fijo (bases de datos, cálculo científico, etc.).

Pero, en general, esta opción no es recomendable, porque no existirá un único tamaño para los archivos, sino muchos tamaños distintos, y esto conduce al poco tiempo a huecos difícilmente aprovechables. A este fenómeno se le llama *fragmentación externa*. Una forma de solucionarla sería que al final de cada secuencia de bloques un puntero indicase el comienzo del siguiente fragmento del archivo, pero esto forzaría un acceso secuencial a los archivos, que es lento e ineficiente.

La forma de solucionar el problema consiste en dividir el espacio disponible en bloques de tamaño fijo, digamos de 512, 1024 u otra potencia entera de 2, y proporcionar algún mecanismo que permita asignar un bloque a un archivo, cualquiera que sea su posición. Es necesario resaltar que la división del espacio en bloques, por sí sola, no excluye la posibilidad del almacenamiento contiguo de archivos, simplemente hace posible la aplicación de técnicas que eviten la fragmentación externa.

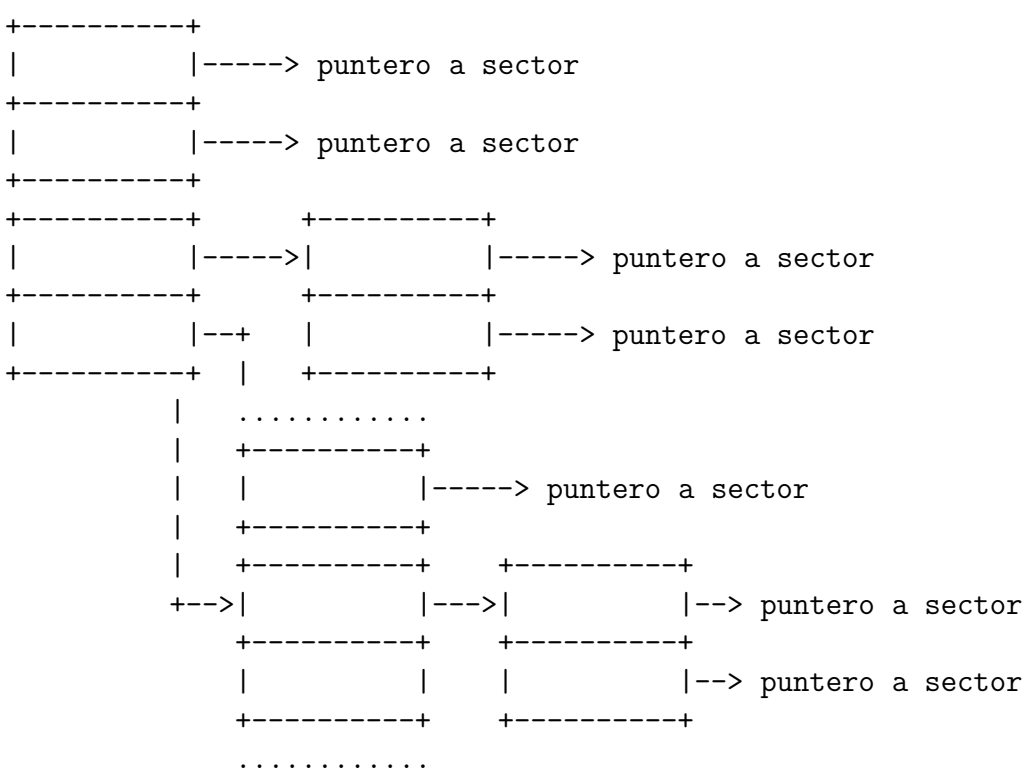
Llegamos por tanto a la conclusión natural de que es más eficiente en términos de espacio usar almacenamiento no contiguo para los archivos. De todas las técnicas que se han implementado, casi todas son variantes de dos estrategias básicas: encadenamiento e indexación.

Hemos comentado anteriormente el encadenamiento como una posible solución, llegando a la conclusión de que esta estrategia fuerza el acceso secuencial a los archivos. Pero, además, prohibe la lectura multisector, más eficiente, pues aún cuando un archivo se encuentre distribuido en sectores contiguos, no existe forma de conocer de antemano esta circunstancia. Además, un puntero dañado puede hacer que miles de bloques queden inaccesibles. No obstante, merece la pena mencionar las ventajas: elimina la fragmentación externa, el espacio necesario para la gestión es pequeño, menos del 1 %, facilita el manejo de bloques defectuosos y la inserción de bloques en un archivo.

La indexación utiliza también punteros para acceder a los bloques pertenecientes a un fichero, pero en lugar de mantener la lista dispersa por todo el disco, se dedican bloques especiales, llamados bloques de índices, para guardar los punteros. De esta forma, una vez leído el bloque de índices, queda eliminada la diferencia entre acceso aleatorio y secuencial. Con bloques de 512 octetos y punteros de 4 octetos, cada bloque de índices podría contener 128 punteros, por lo que el tamaño de los archivos quedaría limitado a 64 KB, inferior a las necesidades habituales. Para remediar esta situación se usan índices multinivel. Esta técnica consiste en que ciertas entradas de los

bloques de índices no apuntan a sectores de archivo, sino a otros bloques de índices. Este esquema puede extenderse a varios niveles. Por ejemplo, UNIX usa indirecciones simples, dobles y triples. Consideremos el caso en que reservamos las dos últimas entradas para indirecciones simple y doble. Entonces, el tamaño mayor de archivo que puede gestionarse es de 8.518.656:

```
126*512+128*512+128^2*512=8,518,656
```



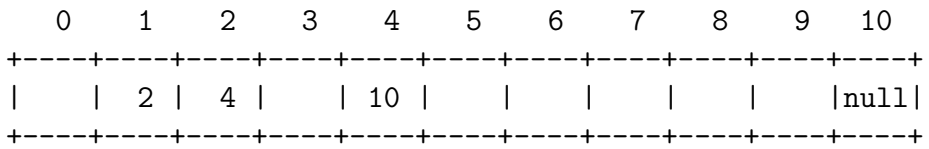
La indexación triple aumenta este límite hasta 1 GB por archivo. Tanto el acceso secuencial como el acceso aleatorio a archivos requiere, mediante esta técnica, una lectura de un bloque de índices por cada nivel de indexación, pero normalmente no es necesario recurrir a indexaciones de alto nivel. Por ejemplo, en mi instalación Linux, la orden

```
find / -size -64k | wc -l
```

arroja un valor de 29.580 archivos, que es el 96 % del número total de archivos. Por tanto, la mayoría de los archivos pueden gestionarse usando un solo bloque de índices.

Falta por hablar del problema que representa saber si un determinado bloque se encuentra libre u ocupado. El método más efectivo consiste en usar un mapa de bits. Un mapa de bits es una secuencia de octetos donde cada bit representa un sector. Asi, si el bit i, que pertenece al octeto $i/8$ y ocupa la posición $i\,\%8$ dentro de ese octeto se encuentra a 1, quiere decir que el sector i se encuentra ocupado. Si el bit se encuentra a cero, el sector se encuentra libre. Consideremos por ejemplo un disco de 2 GB. El mapa de bits ocupa 512 KB, una cantidad que puede matenerse en memoria, y que representa una fracción despreciable del espacio total en disco.

Se ha recurrido a sistemas mixtos, donde el mapa de sectores libres contiene simultáneamente la lista de sectores correspondiente a un archivo. A este sistema se le denomina FAT (File Allocation Table), y su fundamento es el siguiente: se dispone de un vector de punteros. El número de punteros en el vector es igual al número de sectores en el disco. Si el primer sector de un archivo es el sector k, la entrada k del vector apunta al siguiente sector del archivo, sea l. Entonces, la entrada l del vector contiene el sector siguiente, y así sucesivamente. Por ejemplo, un archivo que ocupa los sectores 1, 2, 4, 10 tiene la representación siguiente en la FAT:



El método tiene sin embargo varias desventajas importantes. En primer lugar, el tamaño. Con punteros de 32 bits, un disco de 2 GB requiere 16 MB de espacio para la FAT. Demasiado grande para tenerla toda en memoria. En segundo lugar, el acceso a un archivo requiere disponer de toda la FAT, que contendrá los datos de nuestro archivo y los datos de todos los archivos restantes, que no nos interesan. En tercer lugar, existe mucha distancia entre los punteros y los sectores, lo que requiere movimientos adicionales del cabezal. Por todos estos motivos, éste método está descartado en muchos sistemas.

Finalmente, indiquemos que tanto el encadenamiento como la indexación que hemos expuesto anteriormente no hacen diferencia entre los bloques de disco dedicados a guardar datos y los bloques de disco dedicados a la administración de los primeros (metadatos). Se están actualmente desarrollando sistemas de archivos que distinguen muy claramente entre los datos y los metadatos: reservando espacios distintos, incluso dispositivos físicos distintos

para almacenar unos y otros, y aplicando para su gestión algoritmos diferentes. Se consigue de esta forma un incremento en el rendimiento y una mayor robustez a la hora de restaurar un sistema que ha sufrido una caida.

9.5. Memorias intermedias

Normalmente, un usuario efectúa operaciones de lectura/escritura en un reducido número de archivos. Incluso en entornos multiusuario el número de archivos que se leen o escriben sigue siendo una muy pequeña fracción del número total de archivos contenidos en el disco. Por este motivo, se suele mantener en memoria una reserva de sectores, de manera que cuando una aplicación solicita acceso a un sector determinado, primero se comprueba si dicho sector se encuentra en memoria. Si es así, se traslada al espacio de direcciones del usuario. De la misma forma, una petición de escritura puede hacerse sobre la copia del sector que se encuentra en memoria, y diferir la escritura a disco para un momento posterior, por ejemplo, cuando el proceso que está usando el archivo lo cierra. Esto último encierra el peligro de corrupción del sistema de archivos, por ejemplo, si se produce un corte en la alimentación eléctrica antes de actualizar el disco con los sectores mantenidos en la *cache*. Por eso, a veces se limita la memoria intermedia para operaciones de lectura, hablandose entonces de memorias intermedias de *escritura directa*. De todos modos, como la mayoría de los accesos a disco son de lectura, el impacto sobre el rendimiento es pequeño.

Aunque los detalles de implementación varían entre un sistema y otro, un sistema de *cache* de disco requiere varios elementos:

- espacio en memoria para almacenar una serie de sectores (un vector de sectores). Cuantos sectores es algo que se determina en tiempo de configuración del sistema.
- un mecanismo de indexación que permita, dada una petición de un sector determinado, localizarlo en el menor tiempo posible. Un algoritmo de dispersión puede hacer esto en un tiempo O(1), con un peor caso de O(n).
- un método que permita saber qué entradas del vector están libres y qué entrada ocupadas.
- un mecanismo que permita decidir cuando un sector determinado ya no es preciso mantenerlo en la *cache*.

El sistema de *cache* de disco de Berkeley UNIX elimina el 85 % de los accesos a disco que de otra forma serían necesarios.

Capítulo 10

Unidades de disco (III)

10.1. Intefaces para acceso a unidades de disco

10.1.1. Unidades IDE

En el momento actual, la intefaz con unidades de disco flexible no tiene gran interés, debido esencialmente a que éste es un medio de almacenamiento en retroceso, aunque hay que reconocerle una capacidad de supervivencia excepcional. Los detalles de la programación de la controladora de disco flexible son engorrosos, debido paradójicamente a su simplicidad, que obliga al programador a dedicarse a detalles como el arranque y parada del motor de la unidad, el movimiento del cabezal o la programación del chip de acceso directo a memoria, que es un dispositivo al mismo tiempo complejo y obsoleto. Nosotros no trataremos este tema.

Por lo que respecta a discos rígidos, existen esencialmente dos interfaces: IDE [1] y SCSI.

IDE es la abreviatura de *Intelligent Drive Electronics*. Para comprender la razón de ser de IDE hay que mirar al diseño anterior: un controlador separado de la unidad de disco, que era un elemento pasivo. La separación de los datos de la señal de reloj, el cálculo de las órdenes, la gestión de gran variedad de geometrías de disco, la gestión de sectores defectuosos, etc. era tarea de este controlador. Piénsese en el simple hecho de que la conexión entre disco y controlador tenga que dedicar más de la mitad de su capacidad a datos "inútiles", como señales de reloj y códigos CRC.

[1] también conocida como ATA

De ahí la necesidad de integrar en una misma unidad la parte física y el control electrónico. Por eso surgió IDE.

IDE se presenta como una interfaz lógica entre el sistema y la unidad de disco, y acepta órdenes de alto nivel. Por ejemplo: leer un sector, formatear una pista, etc. Por contra, la interfaz a la que sustituyó era una interfaz física que aceptaba órdenes de bajo nivel, como arrancar el motor y mover el cabezal.

Los discos IDE, con su controladora integrada, están preparados para efectuar la traducción entre sectores lógicos y sectores físicos [2] , lo cual deja libertad a los fabricantes para colocar mayor número de sectores por pista en la parte externa del disco. Además, estos discos gestionan su propia *cache*, lo que permite realizar siempre lecturas multisector y reducir el tiempo medio de acceso.

Volvemos a incidir en que IDE es una intefaz lógica, y por tanto cualquier dispositivo que la satisfaga funcionará correctamente, independientemente de cual sea su construcción. Por ejemplo, usan esta interfaz unidades de disco semi-rígido de alta capacidad (ZIP), unidades ópticas y unidades magneto-ópticas.

10.1.2. Interfaz SCSI

SCSI es un acrónimo de *Small Computers Systems Interface*, y al contrario que IDE, orientado a unidades de almacenamiento, SCSI está diseñado para la conexión de cualquier tipo de dispositivo que satisfaga la intefaz lógica que esta norma define. Pueden conectarse discos rígidos SCSI, pero también impresoras, *scanners*, unidades ópticas, etc.

Como soporte de esta lógica, la norma SCSI define su propio bus, al que pueden conectarse hasta 8 dispositivos. Uno de ellos es un adaptador que conecta el bus SCSI con el bus del sistema, y por tanto las unidades SCSI no dependen de la arquitectura del sistema al que se encuentran acopladas, pues sólo conocen su propio bus y el adaptador, que a su vez es otra unidad SCSI.

[2] realmente, esta es una capacidad de *Enhanced* IDE, que permitió direccionamiento lógico de sectores en lugar del esquema cilindro, cabeza, sector

Todos los dispositivos SCSI son "inteligentes.[en] el sentido en que lo son los dispositivos IDE, pero además reconocen un protocolo que les permite comunicarse entre sí, a través de su propio bus, y sin intervención de la CPU. Por ejemplo, puede efectuarse una copia entre una unidad CD-ROM SCSI y un disco del mismo tipo de forma autónoma.

10.1.3. Serie versus Paralelo

El bus SCSI es un bus paralelo, al igual que lo es la conexión IDE con el bus del sistema. Sin embargo, las últimas implementaciones de los buses serie parece que, a corto plazo, inclinarán la balanza hacia este método de transmisión. Un avance pueden ser las unidades USB, y desde hace algunos meses, la introducción progresiva de SATA: *Serial ATA*. Sin olvidar Ethernet, RapidIO o Firewire, entre otros.

Concretamente, la nueva SATA ofrece de entrada 150 MB/s, y está previsto que alcance los 600 MB/s a medio plazo. A la simplicidad de construcción y velocidad se unen algunas ventajas adicionales de carácter físico. En primer lugar, la transmisión de señales vía serie puede realizarse a distancias mayores sin pérdida de calidad. En segundo lugar, SATA usa señales de 250mV, en contraste con los 5V de IDE. En tercer lugar, se evitan las interferencias entre hilos paralelos, que, cuando se excitan a altas frecuencias, radian energía electromagnética a modo de antenas.

10.2. Acceso a discos mediante BIOS

10.2.1. Estado de la unidad

La interrupción 13h proporciona una serie de funciones para acceder a unidades de disco flexibles. Esta interrupción también contiene funciones para acceder al disco duro, y, generalmente, la llamada es idéntica para discos flexibles o rígidos, indicando simplemente en cada caso el número de la unidad en DL. Así, las unidades de disco flexible tienen los números 0 y 1, mientras que los discos rígidos se identifican por 80h y 81h. Otra cosa en común de estas funciones es la devolución del estado de la unidad en AH, de acuerdo con el código que se recoge a continuación:

Código	Significado
00h	No hay error
01h	Número de función no permitido
02h	No se encontró marca de dirección
03h	Intento de escritura en disco protegido
04h	Sector no se encuentra
06h	Disco se cambió
08h	Rebase DMA
09h	Rebase del limite de segmento
10h	Error de lectura
20h	Error de la controladora
40h	Pista no encontrada
80h	Unidad no responde

El estado por otra parte puede obtenerse explícitamente mediante la funcion 01h, que devuelve la información de la tabla anterior en AH. Antes de la llamada, es preciso colocar en DL el número de la unidad.

Si después de una llamada a una función se recibe un código de error, es conveniente reiniciar la unidad, de lo que se encarga la funcion 00h. Es indiferente si se indica la unidad 0 o la 1, pues esta función reinicia todas las unidades de disco flexible.

Frecuentemente, los programas necesitan saber con que tipo de unidad están trabajando. Para ello se usa la función 08h. Colocando como siempre el número de unidad en DL, se obtiene en AH un codigo de error (0 = no hay error) y en BL el tipo de unidad:

```
BL
01h         5"1/4    320K
02h         5"1/4    1.2M
03h         3"1/2    720K
04h         3"1/2    1.44M

DH      Numero de cara mayor
CH      Numero de pista mayor
CL      Numero de sector mayor
ES:DI   Puntero a DDPT
```

10.2.2. La DDPT

La DDPT (Disk Drive Parameter Table) contiene parámetros que la BIOS necesita para la programación de la controladora. Para cada unidad y para cada formato de disco, existe una de estas tablas en ROM, pero, además, es posible instalar una de estas tablas propias, ya que la BIOS siempre referencia la DDPT actual mediante un puntero lejano que se guarda en las posiciónes de memoria que normalmente ocupa la interrupcion 1Eh. Como esta interrupción no se usa por el hardware del PC, ni por el BIOS, ni por el DOS, es posible usar estas posiciónes de memoria. La DDPT se compone de 11 octetos que codifican parámetros del funcionamiento del disco, como los tiempos necesarios para levantar y posar el cabezal, los tiempos de arranque y de paro del motor, el tamaño del sector o el número de sectores por pista.

10.2.3. Lectura y escritura de sectores del disco

Las funciones de lectura y escritura de sectores individuales son funciones básicas del BIOS. La función de lectura, 02h, requiere saber el número de sectores que se van a leer, pues pueden leerse varios de una vez. Esta información se coloca en AL. En DL se coloca el número de unidad y en DH, CL y CH respectivamente el número de cabezal de escritura (0 = arriba, 1 = abajo), el número de sector y el número de pista. Los datos no son colocados en una zona fija de memoria, sino que es necesario especificar, antes de llamar a la función, la dirección de un *buffer* en ES:BX. Queda prevenido el lector contra la utilización de un *buffer* de tamaño inferior al número de octetos que vayan a leerse. A la vuelta de la función, AH contiene el estado de error y AL el número de sectores leidos.

Para leer el primer sector de una unidad de disco duro, DH tomará el valor 1, en contraposición a unidades de disco flexibles, donde se tomará el valor 0.

La función 03h es la encargada de escribir sectores en discos flexibles. Acepta los mismos parámetros que la 02h, con la lógica excepción de que ahora AL contiene el número de sectores a escribir. Naturalmente, el *buffer* ya ha de estar cargado con la información antes de la llamada a la función.

Como ejemplo, el siguiente fragmento de código lee el primer sector de la primera unidad de disco rígido:

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
main()
{
    struct REGPACK reg;
    int j;
    char *buffer=(char *)malloc(512);

    reg.r_ax=0x0201; /* ah=2, funcion; al=1, leer un sector */
    reg.r_cx=0x0001; /* ch=0, cilindro; cl=1 sector */
    reg.r_dx=0x0080; /* dh=0, cabezal; dl=0x80, primer disco rigido */
    reg.r_es=FP_SEG(buffer);
    reg.r_bx=FP_OFF(buffer);
    intr(0x13,&reg);

    /* imprimir el primer sector */
    for(j=0;j<512;++j){
        printf("%c",*( buffer+j));
        if ((j%26)==0) printf("\n");
    }
    return(0);
}
```

10.3. Acceso directo a la controladora IDE

La programación y ejecución de órdenes en una interfaz IDE se realiza, al igual que en el caso del controlador de disco flexible, en tres etapas:

- Fase de órdenes. La CPU prepara el registro de parámetros y envía a la interfaz el código de la orden
- Fase de datos. El disco coloca sus cabezales, obtiene los datos del disco y los pone a disposición de la CPU
- Fase de resultados. La controladora ofrece el estado de la unidad como resultado de la orden ejecutada. Finalmente, se lanza la interrupción 0x14. Esta interrupción se limita a dar valor a una bandera situada en el segmento 0x40, desplazamiento 0x8e

La escritura de órdenes y la lectura de resultados se realiza a través de varios puertos, y, al contrario que en la controladora de disco flexible, no se usa DMA, sino entrada/salida programada. La sincronización entre la CPU y la controladora se realiza a través de la interrupción 0x14, en varios momentos precisos:

1. Cuando se lee un sectore. Una vez la controladora ha dejado el sector a disposición de la CPU en su *buffer* interno, se genera una IRQ 0x14. Si la lectura es de varios sectores, entonces se produce igual número de interrupciones.

2. Cuando se escriben sectores. Se produce una interrupción cada vez que la controladora está dispuesta a recibir nuevos sectores. Sin embargo, no se produce interrupción para el primer sector, supuesto que la escritura se realiza inmediatamente después de haber enviado la orden. Se produce una última interrupción al inicio de la fase de resultados.

3. En cualquier otro caso, se produce una interrupción al principio de la fase de resultados.

Como en otras ocasiones, os remito al libro "The indispensable PC hardware handbook", página 1185 y siguientes, donde se exponen en detalle todas las órdenes que reconoce la interfaz IDE. No obstante, a modo ilustrativo, presento un pequeño fragmento de código donde se realiza la lectura del primer sector del primer disco.

```
/*
Acceso directo a la controladora de disco. Debe compilarse
con la opcion -2 para habilitar las instrucciones 286/287
*/
#include <stdio.h>
#include <alloc.h>

#define HD_restaura      0x10    /* ordenes */
#define HD_busca         0x70
#define HD_lee           0x20
#define HD_escribe       0x30
#define HD_formatea      0x50
#define HD_verifica      0x40
#define HD_diagnosis     0x90
#define HD_parametros    0x91
```

```
#define HD_principal     0x3f6  /* registros */
#define HD_datos         0x1f0
#define HD_error         0x1f1
#define HD_escribe_p     0x1f1
#define HD_sector_c      0x1f2
#define HD_sector        0x1f3
#define HD_cil_b         0x1f4
#define HD_cil_a         0x1f5
#define HD_dis_cab       0x1f6
#define HD_estado        0x1f7
#define HD_orden         0x1f7

#define HD_noreintentar  1

int lee_sector(int unidad, int cabeza, int cilindro, int sector,
               int operacion, char *direccion, int numsect)
{
    outportb(HD_sector_c,numsect);
    outportb(HD_sector,sector);
    outportb(HD_cil_b,cilindro & 0xff);
    outportb(HD_cil_a,cilindro >> 8);
    outportb(HD_dis_cab, unidad<<4 | cabeza | 0xc0);

    outportb(HD_principal,cabeza & 8);

    poke(0x40,0x8e,0);
    outportb(HD_orden,operacion);
    while (!peek(0x40,0x8e));

    asm{
        push es
        push cx
        push dx
        push di
        mov cx,numsect
        xchg ch,cl
        les di,direccion
        cld
        mov dx,HD_datos
        rep insw
        pop di
```

```
        pop dx
        pop cx
        pop es
    }
    return(0);
}

main()
{
    int j;
    char *buffer=(char *)malloc(512);
    lee_sector(0,0,0,0,HD_lee | HD_noreintentar,buffer,1);
    for(j=0;j<512;++j){
        printf("%c",*(buffer+j));
        if ((j%26)==0) printf("\n");
    }
}
```

Capítulo 11

Otros medios de almacenamiento

11.1. Almacenamiento magneto-óptico

Los sistemas de almacenamiento magneto-optico recurren a una técnica híbrida: puramente óptica para lectura y térmica-magnética para escritura.

En esencia, cuando se desea escribir sobre el sustrato ferromagnético se calienta en un punto mediante un delgado haz láser para elevar su temperatura por encima de la temperatura de Curie, destruyendo la magnetización anterior, y a continuación mediante una bobina cuyo plano es perpendicular al haz láser se induce la nueva magnetización para representar un 0 o 1 binarios.

Para el proceso de lectura, se hace incidir el haz sobre un dominio magnético y se estudia la polarización del haz reflejado. Este cambio de polarización de un haz de luz al reflejarse sobre un dominio magnético se denomina *efecto Kerr*.

En teoría, este método es capaz de superar la barrera impuesta por el efecto superparamagnético. Para ello, los bits se inscriben perpendicularmente a la superficie del disco, en lugar de paralelamente como en los sistemas magnéticos convencionales.

Dos dificultades principales están sin embargo retrasando la adopción de esta tecnología. Por un lado, es preciso enfocar el haz láser sobre un punto muy pequeño. Para ello se recurre a una lente de inmersión en sólido, una

técnica derivada de la microscopía de inmersión en líquido. Por otro lado, la técnica requiere que la distancia del sistema de enfoque a la superficie del disco sea muy pequeña, del orden de la longitud de onda del láser.

Actualmente, esta técnica está siendo desarrollada por la empresa Terastor, con participación de Quantum y acuerdos con Olympus para los componentes ópticos y HP para la electrónica de apoyo.

11.2. Almacenamiento óptico y holográfico

Otra posibilidad la constituyen los medios de almacenamiento ópticos, con discos CD y discos DVD. La densidad de almacenamiento varía entre los 0.11 Gigabytes/cm^2 en los discos CD y los 0.51 Gigabytes/cm^2 para los discos DVD. Aunque existen diferencias de implementación que justifican estas distintas capacidades de almacenamiento, la técnica básica es idéntica. Los bits se codifican realizando pequeñas muescas sobre una pista espiral que comienza en la parte interior del disco y acaba cerca del borde externo. El sustrato es cubierto por una capa de aluminio y una segunda capa de laca protectora.

Para efectuar la lectura, un fino haz láser se desplaza a lo largo de la pista. Cuando encuentra una muesca, cuyo tamaño es del orden de la longitud de onda del láser, se produce difracción que hace que la intensidad de la luz reflejada disminuya drásticamente. El análisis de la intensidad reflejada por el disco permite reconstruir el patrón de bits que fué codificado mediante las muescas.

Por su parte, el almacenamiento holográfico recurre a una técnica radicalmente diferente, basada en la propiedad de algunos materiales para fijar mediante algún tipo de alteración química o física una figura de interferencia de luz. En la técnica holográfica se hacen incidir simultáneamente sobre el medio de registro dos haces láser. Para generarlos se usa un único láser que es dividido en dos mediante un divisor de haz. A continuación uno de los haces resultantes se hace incidir directamente sobre el medio de registro, mientras que el segundo se modifica mediante un modulador espacial de acuerdo con la información contenida en una página de datos. De la interferencia de ambos haces surge un holograma, que queda registrado. La lectura se efectúa iluminando el holograma mediante el haz de referencia. Como resultado, se reconstruye el haz objeto (aquel que fué modulado antes de incidir sobre el

medio de registro). Finalmente, el análisis del haz objeto permite reconstruir la página de datos que se usó para modularlo.

Dos características hacen interesante el almacenamiento holográfico. Por un lado, existe la posibilidad de multiplexar muchos hologramas, es decir, muchas páginas de datos, sobre el mismo medio de registro. Esto se consigue variando los ángulos de incidencia de los haces de registro, sus longitudes de onda, o ambos. Por otro lado, los datos se leen o escriben en paralelo. Los prototipos actuales pueden efectuar un acceso aleatorio en unos 100 μs, mientras que el tiempo de acceso en un medio magnético es del orden de milisegundos.

La investigación en almacenamiento holográfico se encuentra activa desde hace más de dos décadas, y recibe el esfuerzo de compañias como IBM, Rockwell, Lucent o Bayer. Se pueden distinguir dos aproximaciones distintas en el esfuerzo por conseguir un sistema de almacenamiento holográfico práctico. Por un lado, se hacen esfuerzos para basar la producción en componentes que puedan fabricarse en masa, aunque sea a costa de prestaciones más modestas. Por otro, se trabaja en dispositivos donde la prioridad es el tiempo de acceso mínimo a los datos, lo que a su vez puede implicar una reducción en la densidad máxima admisible. En febrero de 2004, la compañia de telefonía móvil japonesa NTT anunció la creación de un prototipo de memoria holográfica sobre un medio de registro estratificado de unos dos centímetros cuadrados, con capacidad para 1 Gigabyte. Se espera su comercialización a lo largo de 2005. Por su parte, la compañia Aprilis comercializa ya una unidad de disco holográfica con capacidad para 200 GB, y anuncia la disponibilidad en 2007 de un disco de 1.6 TB.

11.3. Almacenamiento con resolución atómica

El almacenamiento con resolución atómica es una técnica de campo próximo mediante la cual se altera alguna propiedad del sustrato mediante una corriente de electrones emitida por una punta de tamaño atómico.

En la actualidad, el dispositivo más avanzado es obra de HP. Se trata de un prototipo de unas decenas de milímetros cuadrados capaz de almacenar 2 Gigabytes. Este dispositivo usa como sustrato un material bifásico, con una fase amorfa y una fase cristalina. Ambas fases son estables a temperatura

ambiente. Cuando una punta atómica próxima emite un haz de electrones, induce un cambio de fase, permitiendo la escritura de un bit. Para la lectura, puede emplearse una técnica de campo próximo emitiendo un haz débil de electrones y detectando el cambio en alguna propiedad eléctrica dependiente de la fase. También puede emplearse una técnica de campo lejano, mediante lectura óptica.

Evidentemente, la viabilidad del dispositivo depende de la disponibilidad de microactuadores que permitan posicionar la matriz de agujas con una precisión de unos pocos nanometros. HP ha patentado un micromotor que consigue una precisión de tres nanometros. El último problema consiste en proporcionar al dispositivo un vacio para su funcionamiento, o al menos una atmósfera controlada que evite la dispersión de los electrones, y un continente robusto que permita integrarlo en dispositivos móviles. Con todo, HP espera tener en el mercado uno de estos dispositivos durante 2006, con una capacidad del orden de 10 Gigabytes. En competencia con los medios magnéticos convencionales, una ventaja de estos dispositivos consiste en su bajo consumo de energía: es nulo mientras el dispositivo no funcione, al contrario que un Microdrive, u otro disco convencional, que ha de mantenerse en rotación constante.

11.4. Dispositivos micro-electromecánicos

Los dispositivos micro-electromecánicos pueden considerarse como una reedición de las viejas tarjetas perforadas. El desarrollo pionero en este campo corresponde a IBM y su *milpiés*, desarrollado en el centro de investigación de esta compañia en Zurich. En esencia, se trata de una matriz de nano-agujas capaz de leer y escribir sobre una la superficie de un polímero. El fundamento del dispositivo consiste en que, debido al tamaño microscópico de la punta de cada aguja, al pasar por ella una pequeña corriente eléctrica se eleva su temperatura hasta varios cientos de grados. De esta forma, la aguja deja un pequeño agujero sobre el polímero.

Para leer, se hace pasar una corriente pequeña por la aguja para incrementar su temperatura, pero por debajo de la que se alcanza en la operación de escritura. Cuando la aguja así calentada cae en uno de los agujeros previamente grabados, aumenta bruscamente su superficie de contacto con el polímero, y por tanto desciendo también bruscamente su temperatura, lo que tiene un efecto mensurable sobre la resistencia eléctrica de la aguja.

Finalmente, el equipo de Zurich ha conseguido un polímero especial con la propiedad de que al calentar uno de los agujeros grabados previamente a una temperatura específica, la tensión superficial lo hace desaparecer. Una matriz de agujas, bajo la cual se mueve una superficie de polímero, puede escribir y leer datos en paralelo. Respecto a los medios de almacenamiento convencionales tiene la ventaja de que el dispositivo no consume energía mientras no se efectúe una operación de lectura o escritura. También, puesto que cada aguja accede a una pequeña zona bajo ella, el tiempo de búsqueda se hace muy pequeño, y el tiempo de latencia desaparece.

Capítulo 12

Dispositivos de visualización

12.1. Introducción

Este capítulo trata sobre adaptadores gráficos, y está estrechamente basado en el libro de Hans-Peter Messmer, capitulo 35, *The indispensable PC hardware book* (Addison- Wesley, tercera edición, aunque hay disponible una cuarta).

Existe variedad en adaptadores gráficos, desde simples tarjetas monocromas que sólo pueden mostrar texto hasta rápidas tarjetas de alta resolución para aplicaciones profesionales de CAD, con procesadores dedicados. Una discusión de todos estos aspectos es prácticamente imposible, por lo que nos limitaremos a los conceptos e implementaciones básicos.

12.2. Formación de las imágenes en el monitor

El método usado para mostrar una imagen en un monitor es similar al usado en una pantalla de televisión normal. Un tubo de rayos catódicos genera un haz de electrones, que son acelerados por un ánodo. La trayectoria de este haz puede modularse mediante dos deflectores en dos direcciones perpendiculares. Al impactar contra la pantalla del monitor, recubierta por una sustancia fosforescente, ésta se ilumina.

La pantalla se encuentra dividida en un conjunto grande de líneas horizontales, y cada línea a su vez se encuentra formada por puntos llamados *pixels* (

del inglés *picture elements*). Cuando el haz de electrones ha alcanzado el extremo derecho de una fila, ha de volver al principio de la fila siguiente. A este movimiento se le llama *retraza horizontal*. De la misma forma, cuando se alcanza el extremo derecho de la última línea de la pantalla, el haz debe volver al extremo izquierdo de la primera línea, esquina superior izquierda. A este movimiento se le llama *retraza vertical*. Para mostrar una línea en pantalla, consistente en un conjunto de *pixels*, la intensidad del haz, no sólo su dirección, ha de modularse adecuadamente. En aquellos puntos donde impacte un haz intenso, tendremos un *pixel* iluminado, y en aquellos donde la intensidad del haz sea casi nula, tendremos un *pixel* oscuro. Debido a la persistencia de la fosforescencia en la pantalla, y a la persistencia de las imágenes en el ojo humano, tenemos la sensación de ver imágenes estables, cuando en realidad toda la pantalla se está re-escribiendo entre 50 y 100 veces por segundo, que es la frecuencia a la que se produce la retraza vertical.

Con el haz de electrones descrito, sólo pueden mostrarse imágenes monocromas, con escala de grises. La pantalla de este tipo de monitores está recubierta de un material que emite en verde, naranja o blanco. Para mostrar imágenes en color son precisos tres haces, cada uno de los cuales ilumina puntos de un color distinto: rojo, verde o azul. Mediante mezcla aditiva de estos tres colores puede conseguirse cualquier otro color. Por consiguiente, en los monitores a color los tres haces han de ser modulados. La intensidad total de los tres determina el brillo, y la intensidad relativa entre ellos el color resultante.

Está claro que la modulación del haz, tanto en dirección como en intensidad, ha de estar sincronizada de tal forma que, por ejemplo, la retraza horizontal suceda exactamente cuando se ha alcanzado el extremo derecho de una línea. El adaptador gráfico ha de proporcionar las señales necesarias para conseguir cada *pixel* de la imagen, así como el sincronismo de las retrazas horizontal y vertical.

Consideremos por ejemplo un monitor VGA con una resolución de 640x480 *pixels*. Cada línea tiene 640 *pixels*, y existen 480 líneas. Por tanto, cada 480 retrazas horizontales ocurre una retraza vertical. Si la imagen es construida 60 veces por segundo, la retraza vertical ocurre cada 16.7 ms. Un segundo ejemplo: a una resolución de 1024x768 puntos, con una retraza vertical cada 16.7 ms., ¿ cada cuanto tiempo se produce una retraza horizontal ?. Cada 16.7/768 ms = 0.0217 ms. Cada segundo han de mostrarse 1024*768*1000/16.7 = 47091737 *pixels*.

Los monitores LCD funcionan de manera similar, sólo que no existe ningún haz de electrones. En lugar de eso, cada *pixel* puede ser direccionado individualmente (ver más adelante en este tema). Por tanto, no tienen sentido las retrazas.

12.3. Factores ergonómicos, visibilidad y colores

12.3.1. Agudeza visual

En cualquier caso, monocromo o color, es necesaria una comprensión siquiera somera de los mecanismos de la visión para saber que se le puede y que no se le puede exigir a un monitor, y esto no sólo desde la perspectiva técnica, sino por la importancia que tiene a medio y largo plazo una correcta visualización para personas que han de pasar mucho tiempo mirando un monitor.

Consideremos en primer lugar el problema del mínimo visible. Para ello imaginemos un círculo blanco sobre fondo negro. ¿ Cual es el tamaño mínimo necesario para poder percibirlo ? En teoría, no existe tal tamaño mínimo, pues una fuente puntual será visible siempre que de ella parta un flujo de energía adecuado. Ahora bien, aún considerando al ojo un sistema óptico perfecto, sabemos que la imagen de una fuente puntual no es un punto sobre la retina, sino una mancha de difracción provocada por la apertura finita de la pupila. Esta mancha de difracción provocada por una fuente puntual puede tomarse como el tamaño mínimo perceptible, o dicho de otro modo, éste sería el tamaño máximo de un objeto para que pudiese considerarse como un punto luminoso. Si en lugar de un disco blanco sobre fondo negro consideramos un disco negro sobre fondo blanco, el fenómeno es radicalmente distinto, ya que, debido a la difracción, aparecerá luz en el centro de la imagen que se forme en la retina, y este efecto se incrementará a medida que disminuya el diámetro de la pupila. Se estima que, en condiciones de luz diurna, el diámetro mínimo de un disco para que sea visible es de 25 segundos de arco. En estas condiciones, además, la visibilidad del disco es independiente de su forma. A esto se llama *efecto de sumación*. Consideremos ahora dos puntos luminosos separados e imaginemos que vamos aproximándolos de forma que, en un momento dado, sólo percibimos un punto luminoso. Según la teoría de la difracción, los dos puntos aparecen como imágenes distintas si su separación angular es del orden de $1{,}22\lambda/D$ radianes, donde λ es la longitud de onda de la luz que se esté utilizando y D es el diámetro de la pupila. Si aplicamos este resultado

a una pupila de 5mm de diámetro, para una longitud de onda de 500 nm, vemos que la separación angular mínima teórica es de aproximadamente 25 segundos de arco, aunque en la práctica se mide del orden de un segundo de arco. Como ejemplo, consideremos la máscara de 0.28mm de la mayoría de los monitores, observada a una distancia de cincuenta centímetros. La separación angular es entonces de un segundo de arco, lo que significa que en estas condiciones, los objetos mostrados en pantalla se observan aproximadamente como continuos. Estos límites son muy variables dependiendo de qué es lo que se esté visualizando. Por ejemplo, si repetimos el experimento anterior con barras verticales que se aproximan, en lugar de puntos, se obtiene una separación, y en casos particulares, como el de tratar de determinar si dos trazos esta desalineados, el poder de discriminación del ojo es excepcional, llegando a unos pocos segundos de arco. La explicación de todos estos fenómenos tiene que ver no sólo con las aberraciones y el poder separador del ojo, considerado como instrumento óptico, sino además con la estructura granular de la retina y la distribución sobre la misma de los receptores. Otro factor recientemente descubierto es el de las microfluctuaciones que sufre el cristalino para conseguir aumentar la agudeza visual. Parece ser que este elemento puede ser estimulado de forma automática (inconscientemente para el sujeto) para que altere su forma, incluso perdiendo su simetría de revolución, para aumentar puntualmente la agudeza visual del ojo. Existiría, según esta teoría, una realimentación entre el cerebro y el cristalino responsable de que el ojo presente una agudeza visual superior a la teóricamente posible.

12.3.2. Imágenes fluctuantes

Cuando una fuente luminosa aparece y desaparece en un tiempo muy corto, el ojo percibe siempre una duración mayor del fenómeno, que en ningún caso baja de 0.15 segundos, aproximadamente. A esto se le llama "persistencia". Por otra parte, se ha observado que cuando a un individuo se le presenta una fuente luminosa fluctuante, a partir de cierta frecuencia ya no observa el parpadeo que en realidad se está produciendo, sino una luminancia constante igual a la media temporal. La frecuencia a la que esto sucede se denomina "frecuencia crítica de fusión", y es de la forma

$$fcf = a \log(L) + b \tag{12.1}$$

donde L es la luminancia media temporal y a y b son constantes que habrá que determinar experimentalmente en cada caso. De cualquier forma, la fcf depende de la luminancia, de forma que es del orden de 3 o 4 a bajas luminancia y puede subir hasta cien para luminancias altas. Si usted cree

que su monitor parpadea en exceso, la solución quizás sea tan simple como disminuir el brillo de la pantalla.

12.3.3. Colores

Los colores están adquiriendo una importancia fundamental. No sólo porque la informática de consumo se hace "en colores", sino por la gran cantidad de aplicaciones técnicas que necesitan un control razonable del color: diseño gráfico e industrial, artes gráficas, fotografía digital, etc. Por eso en algunas aplicaciones no es suficiente con que un monitor ofrezca *buenos* colores, sino que se exige además que la calidad de los colores se pueda cuantificar y modificar. Piénsese por ejemplo en la cantidad de colores naranja distintos que se pueden discernir. ¿ Como se puede asegurar entonces que el color naranja que yo veo sobre el monitor sea el mismo color naranja que finalmente obtengo en una impresora ?. Una primera cuantificación de la calidad del color la aportan las cantidades llamadas *Temperatura de distribución* y *Temperatura de color*. Para muchos sólidos incandescentes y llamas, sucede que la distribución espectral de la radiación que emiten es similar a la distribución espectral de la radiación de un cuerpo negro [1] que se encontrase a una temperatura T_d. La temperatura T_d del cuerpo negro, cuando se cumple este supuesto, se llama *Temperatura de distribución* del otro cuerpo. Por otra parte, se le llama *Temperatura de color* de un cuerpo a la temperatura que tendría que tener un cuerpo negro para que presentase el mismo color que el cuerpo en cuestión. Estas magnitudes, con ser usadas habitualmente, son insuficientes para cuantificar el color, que según algunos se define como aquellas características de la luz distintas de sus inhomogeneidades espaciales y temporales. Tradicionalmente se han distinguido las propiedades de claridad, tono y saturación en el color. La primera está relacionada con la luminancia del objeto, la segunda con la longitud de onda, es decir, es la propiedad que permite distinguir un rojo de un verde, y finalmente, la saturación está relacionada con la proporción de blanco. Sin embargo, estas características no permiten identificar cuantitativamente con cierta precisión un color, y por ello se desarrolló un álgebra del color que permite especificar cada color por sus coordenadas en un plano. Este álgebra se basa en el siguiente experimento. Si sobre la mitad de un campo circular enviamos un determinado flujo de luz de un determinado color C, podemos hacer que la otra mitad presente el mismo color enviando flujos de luz l, m, n correspondientes a los colores Rojo, Verde y Azul, de forma que podriamos escribir:

[1] El *cuerpo negro* es una idealización usada en Física

$$c_C = l_R + m_V + n_A \tag{12.2}$$

Ahora bien, no siempre puede obtenerse un color como mezcla de colores simples, pero si un color no puede conseguirse, siempre es posible conseguir el resultado de sumar a ese color una cierta cantidad de uno de los colores primarios, con lo cual:

$$c_C + l_R = m_V + n_A \tag{12.3}$$

o bien

$$c_C = -l_R + m_V + n_A \tag{12.4}$$

y esta es la forma en que deben interpretarse términos negativos en la ecuación anterior, conocida como ecuación tricromática. Si normalizamos la ecuación tricromática a la suma $l + m + n$, obtenemos

$$1 = r + v + a \tag{12.5}$$

La ecuación anterior es la base para la representación gráfica de las coordenadas del color. Aunque podría tomarse un espacio tridimensional donde cada punto representase un color, se prefiere acudir a representaciones bidimensionales, basadas en la propiedad del triángulo equilátero de que la suma de las distancias de un punto a los tres lados es igual a la altura. (Ver el libro de Justiniano Casas *Optica*, pag. 384). En la práctica, y puesto que la suma de las tres componentes ha de valer la unidad, especificadas dos de ellas queda determinada la tercera, y puede acudirse a una representación cartesiana.

12.4. Pantallas planas

La primera de las tecnologías viables para la producción controlable de matrices luminosas fue la de las pantallas de plasma, a finales de los años sesenta. Estos dispositivos constan de dos láminas de vidrio entre las que se encuentra una mezcla de gases (neón entre ellos). Las superficies de las dos láminas se encuentran cruzadas por líneas conductoras paralelas, de manera que las líneas de ambas láminas son perpendiculares entre sí. Cuando se aplica un voltaje suficientemente elevado a una de las intersecciones, se produce una corriente eléctrica momentánea que descompone el gas en iones y electrones,

produciéndose emisión de luz. Puesto que la emisión se produce durante un periodo muy corto de tiempo, la luminosidad obtenida es baja, aunque se consigue mejorarla si se controla el voltaje en las intersecciones mediante corrientes alternas. Uno de los inconvenientes de este tipo de pantallas es la dificultad para conseguir imagenes en color (la mayoría son de color naranja). El otro, como se ha dicho, es la baja luminosidad. El primero puede superarse mediante el recubrimiento con fósforo de distintos colores. Es decir, en cada intersección se produce luz de un color determinado al excitarse un compuesto específico.

El fenómeno de la electroluminiscencia también se ha usado en la construcción de pantallas planas. En esta técnica, una sustancia fosforescente, que suele ser sulfuro de zinc dopado con manganeso, se coloca entre dos capas aislantes que contienen electrodos ortogonales. Cuando el voltaje en una intersección supera un valor umbral, se produce una corriente que descompone el sulfuro de zinc, produciéndose una corriente eléctrica que excita a los átomos de manganeso, los cuales emiten luz de color amarillo.

Tanto las pantallas de plasma como las de electroluminiscencia son pantallas de *emisión primaria*, es decir, la luz se produce en la misma pantalla. Una técnica radicalmente distinta usan las pantallas de cristal líquido.

Un cristal líquido es una sustancia cuyas moléculas pueden moverse independientemente unas de otras, pero que, por efecto de débiles fuerzas intermoleculares, tienden a alinearse de forma que existe una dirección predominante. Una de las familias de cristales líquidos son los llamados *cristales nemáticos*, cuyas moléculas tienen forma alargada. La orientación de los cristales nemáticos puede conseguirse mediante la exposición a un campo eléctrico o colocándolos en las proximidades de ciertas superficies. A su vez, la orientación de las moleculas nemáticas determina las propiedades ópticas de los cristales, y en particular su coeficiente de transmisión.

La configuración típica de una pantalla plana de cristal líquido consiste en dos láminas polarizadoras cruzadas entre las cuales se encuentra el cristal líquido. Ocurre que las moléculas nemáticas próximas a la superficie del polarizador tienden a orientarse según la dirección de polarización, de modo que si nos movemos desde la superficie del primer polarizador hasta la superficie del segundo, las moléculas nemáticas rotan 90 grados. Como el plano de polarización de la luz gira junto con los cristales, si se ilumina por detrás la pantalla, la luz se transmitirá. Ahora bien, si se somete al cristal a un campo

eléctrico perpendicular a las superficies de los polarizadores, las moleculas se alinearán en esta dirección, y la luz que pase por el primer polarizador no podrá atravesar el segundo. De esta forma puede controlarse qué *pixels* transmiten y qué *pixels* no lo hacen. Llamamos a estas pantallas de *matriz pasiva*.

El inconveniente de este tipo de pantallas es el conflicto entre resolución y contraste. Cuando aumenta la resolución, la diferencia de voltaje entre una intersección activa y otra que no lo está puede reducirse a unas pocas unidades por ciento, cuando imágenes bien contrastadas precisan una diferencia en torno al 50 %. Se usan fundamentalemente dos técnicas para aumentar el contraste al tiempo que la resolución. La primera consiste en usar cristales nemáticos de supertorcedura. Es decir, cristales que giran más de 180 grados entre los dos polarizadores. De esta forma se consigue aumentar la pendiente de la curva transmisión-voltaje. Es decir, un incremento en el voltaje se traduce en un incremento mayor en la transmisión.

La segunda técnica da lugar a las llamadas pantallas de matriz activa. Cada *pixel* viene controlado por un transistor, que se ocupa de mantener constante el voltaje en tanto que no se reciba una nueva señal de control. De esta forma pueden conseguirse altas resoluciones con contrastes muy satisfactorios. A estas pantallas se les denomina TFT (del inglés *Thin Film Transistor*, película delgada de transistores)

12.5. Estructura general de los adaptadores de video

En su estructura general, los adaptadores gráficos son todos parecidos, a pesar de las diferencias en cuanto a colores y resolución. La parte central del controlador de video es el CTRC (controlador del tubo de rayos catódicos), que supervisa el funcionamiento del adaptador y genera las señales de control precisas.

La CPU accede a la RAM de video a través del interfaz del bus del sistema para escribir la información que define el texto o gráfico que ha de ser representado en pantalla. Por ejemplo, en un adaptador VGA en modo texto, color, 80 columnas por 25 filas, la información que se representará en pantalla se escribe a partir del desplazamiento 0 del segmento $0xb800$. El

CTRC genera contínuamente direcciones de RAM de video para leer los caracteres correspondientes, y para trasladarlos al generador de caracteres. En modo texto, los caracteres se definen por sus códigos ASCII, con un atributo asociado que determina los colores, parpadeo e intensidad. Para cada código ASCII, se guarda en ROM un patrón de bits con la representación del carácter. Por ejemplo, el patrón para la letra A es similar a este:

```
00000000
00111100
01100110
01100110
01100110
01100110
01100110
01111110
01111110
01100110
01100110
01100110
01100110
01100110
01100110
00000000
```

y como puede verse se compone de 16 octetos. Por tanto, en el mapa de caracteres, el caracter x se encuentra en la posición $16x$. Puede haber más de un mapa de caracteres, correspondientes a varias fuentes, por ejemplo, 8*8, 8*14 y 8*16. Los patrones de bits para carácter son transformados en una corriente de bits, que pasa a un registro especial. Un generador de señales lee esta corriente, y junto con la información sobre atributos produce la señal adecuada que modula el haz de electrones. En modo gráfico, la información de la RAM de video se usa directamente para generar la pantalla. Es decir, es innecesario el generador de caracteres. Por ejemplo, usando un octeto por cada *pixel*, con una resolución de 640*480, se necesitan 307200 octetos, uno por *pixel*, y cada uno de ellos se puede representar con 256 valores distintos, correspondientes a 256 colores distintos o valores de gris. Si se usan sólo dos bits por *pixel*, se tendrán sólo 4 posibles colores o grises, pero toda la pantalla quedará descrita con 76800 octetos.

Volvamos a la generación de caracteres en modo texto. Cada carácter en pantalla tiene asignados dos octetos en RAM: carácter y atributos. El genera-

dor de señales del adaptador gráfico muestra el carácter de acuerdo con su atributo: colores, vídeo inverso, parpadeo e intensidad. Esta es la estructura de la palabra correspondiente a un carácter:

```
+---+---+---+---+---+---+---+---+
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  primer octeto: codigo ASCII
+---+---+---+---+---+---+---+---+
+---+---+---+---+---+---+---+---+
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  segundo octeto: atributos
+---+---+---+---+---+---+---+---+
  |  -----+----   |  -----+----
  |       |       |       |
  |       |       |       +--> color texto
  |       |       +----------> intensidad
  |       +------------------> color fondo
  +--------------------------> parpadeo
```

Como se ha dicho, el código ASCII del carácter indexa una tabla en ROM donde se encuentran los patrones de bits de los caracteres. Existe sin embargo una diferencia entre la representación en ROM y en pantalla. En esta última se añade una novena columna para separar un carácter del adyaccente. Además, este noveno *pixel* se ilumina para ciertos caracteres que requieren continuidad, como la línea horizontal. Es posible conmutar entre la representación de 9 columnas y la representación de 8 columnas (ver Tischer). El siguiente programa llama a una interrupción de la BIOS que devuelve la dirección de la tabla 8*16 en ROM, los convierte en cadenas de ceros y unos y los guarda en un archivo, de manera que puedan examinarse cómodamente:

```
/*

Este programa lee los patrones de la tabla de caracteres
y los pone en un buffer en RAM, donde pueden modificarse.
Tambien se salvan a disco, para que puedan ser examinados.

+--------------+
| tabla en ROM |<-------p
+--------------+        q=p..p+4096

Compilese con Turbo C o Turbo C/C++, con cualquier version.

*/
```

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include <stdlib.h>

main(int argc, char *argv[], char *envp[])
{
  struct REGPACK rp; /* para llamar a la interrupcion*/
  char *p,*q;        /* para apuntar a ROM */
  char *m,*n;        /* para mi buffer de manipulacion*/
  FILE *f;           /* archivo para los datos */
  int i,j,k;         /* contadores genericos */
  char cadena[9];    /* descripcion 0/1 de cada fila */

  /* llamar a la interrupcion*/
  rp.r_ax=0x1130;
  rp.r_bx=0x0600;
  intr(0x10,&rp);

  /* apuntar a la tabla ROM */
  p=(char *)MK_FP(rp.r_es,rp.r_bp);

  /* extraer los datos */
  q=p;
  if ((f=fopen("patrones.dat","w"))==NULL) return(1);
  for(i=0;i<256;++i){       /* para cada caracter */
    for(j=0;j<16;++j){       /* para cada fila */
      for(k=0;k<8;++k){      /* para cada columna */
        if (((*q)&(128>>k))==(128>>k))
          cadena[7-k]='1';
        else
          cadena[7-k]='0';
      }
      cadena[8]=' ';
      fprintf(f,"%s\\n",cadena);
      ++q;
    }
    fprintf(f,"\\n");
  }
  fclose(f);
```

```
}
```

Por ejemplo, considérese la palabra `HOLA` escrita en la primera línea de la pantalla, en modo texto. La representación de esta palabra es:

```
00000000 00000000 00000000 00000000  --> primera linea de rastreo
11000110 00011000 11000000 00011110  --> segunda linea de rastreo
11000110 01100110 11000000 00110011
11000110 01100110 11000000 00110011
11000110 01100110 11000000 00110011
11000110 01100110 11000000 00110011
11111110 01100110 11000000 00110011
11111110 01100110 11000000 00111111
11000110 01100110 11000000 00111111
11000110 01100110 11000000 00110011
11000110 01100110 11000000 00110011
11000110 01100110 11000000 00110011
11000110 01100110 11000000 00110011
11000110 01100110 11000001 00110011
11000110 01100110 11111111 00110011
11000110 00011000 11111110 00000000 --> linea 16 de rastreo
```

Después de cada retraza horizontal se incrementa un registro especial, y por tanto se pasa a la siguiente línea. Después de 16 retrazas horizontales se ha completado la primera línea de la pantalla, y se incrementa un segundo registro que direcciona la segunda línea de texto.

12.6. Estructura de la RAM de video

La RAM de video está organizada de forma diferente, dependiendo del modo de video y el adaptador gráfico usado. En adaptadores gráficos con hasta 128KB de RAM, puede accederse a toda la RAM de video vía CPU. Por ejemplo, en modo 320x200 con 256 colores son precisos 64KB de RAM, mientras que a 640x480 con 16 colores son precisos 150KB para almacenar una pantalla, y por tanto los 128KB de la ventana de video son insuficientes. Por eso, las tarjetas (S)VGA con más de 128KB de RAM implementan un selector que puede manipularse por software y que permite acceder a varias ventanas de 128KB sobre la memoria de video total, residente en la propia

tarjeta. Cómo acceder y manipular estos selectores no se encuentra normalizado, de manera que depende de cada fabricante. Si no se dispone de esta información, es preciso acudir a la BIOS. Actualmente, las tarjetas SVGA conformes VESA sí que comparten un método para acceder y cambiar el selector.

En modo texto, la RAM de video es un arreglo lineal, donde cada palabra de dos octetos guarda un carácter y sus atributos, según el esquema que se explicó anteriormente. Puesto que en modo texto los requerimientos de RAM son escasos (4000 octetos por pantalla de 80x25), existen en realidad varias páginas dentro del segmento 0xb000 (monocromo) o 0xb800 (color), con desplazamientos 0x0000, 0x1000, 0x2000, etc. La BIOS del PC esta preparada para manejar hasta 8 páginas, y conmutar entre ellas.

En modo gráfico, la situación es más complicada. Por ejemplo, en las tarjetas Hércules (las posteriores guardaron compatibilidad con ésta) la RAM estaba dividida en cuatro bancos por página. El primer banco acomoda los *pixels* para las líneas de rastreo 0,4,8..., el segundo, los *pixels* para las líneas 1,5,9..., el tercero, las líneas 2,6,10... y finalmente el cuarto las líneas 3,7,11.... A su vez, los 64KB de vídeo de la tarjeta Hércules se dividen en dos páginas de 32KB:

```
   pagina 0           pagina 1
+------------------|------------------+
+----+----+----+----|----+----+----+----+
   bancos 0,1,2,3      bancos 0,1,2,3
```

Como se ha dicho, la CPU puede acceder a la RAM de video, pero al mismo tiempo la tarjeta gráfica ha de acceder también. ¿Qué ocurre si la CPU pretende leer al tiempo que la tarjeta está escribiendo? En los primeros adaptadores gráficos para PC no existía protección frente al acceso simultáneo, lo que daba lugar a efecto nieve, particularmente en los adaptadores CGA. Esto hacía que sólo se pudiese escribir en RAM de video durante el breve espacio que duraba el periodo de retraza vertical. En los adaptadores posteriores, este problema fue solucionado bloqueando el acceso de la CPU y el adaptador gráfico al mismo banco. Entonces, la CPU tenía que insertar más o menos ciclos de espera. Una solución mejor se implementó poco después, y consistió en el uso de memoria de puerta dual, lo que aseguraba accesos simultáneos sin conflicto. Digamos como curiosidad que existen muchos juegos diseñados para actualizar la pantalla sólo durante el periodo de retraza que

pueden seguir usándose aun cuando las CPU han incrementado su velocidad en dos órdenes de magnitud. Sin embargo, juegos más modernos dejan de poder usarse al aumentar la frecuencia del procesador.

En lo que sigue, centraremos la exposición en las tarjetas VGA.

12.7. El adaptador VGA

12.7.1. Características generales

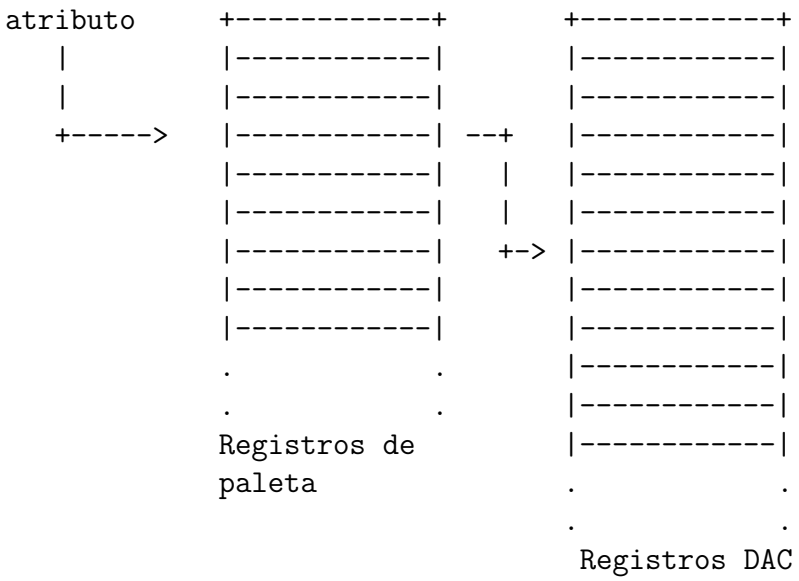
El adaptador VGA (Video Graphics Array) fué introducido por IBM para sus modelos PS/2, y representó un gran avance frente a modelos anteriores. La principal característica no fué el aumento de resolución a 640x480 *pixels*, sino la extensión de la paleta de colores a 262.144, de los cuales pueden representarse en pantalla simultáneamente 256. Para conseguirlo, el monitor VGA no es digital, sino analógico. Estas son las características básicas del adaptador VGA:

- Segmento de video: 0xb000 (texto), 0xa000 (gráfico)
- Video RAM: 256KB
- páginas de pantalla: 1-8
- puertos de E/S: 0x3b0, 0x3df
- matriz de caracteres: 9x16
- resolución: 640x480
- colores: 256 simultáneos
- frecuencia horizontal: 31.5KHz
- frecuencia vertical: 50-70Hz
- ancho de banda: 28MHz
- BIOS propia: Sí

En la VGA, todos los registros pueden leerse, lo que da oportunidad a los programas para informarse del modo de video actual y parámetros de funcionamiento. Por ejemplo, la BIOS VGA implementa la función 0x1c, que guarda y restaura el modo actual, lo que es importante en entornos multitarea, donde un programa puede alterar el modo de video existente. La generación del color es controlada mediante 256 registros en un conversor digital/analógico. Cada registro se divide en tres grupos de 6 bits. El primer grupo especifica el rojo, el segundo el verde y el tercero el azul. Esto proporciona 2^{18} colores posibles. Pero como el DAC tiene 256 registros, solo 256 colores pueden mostrarse simultáneamente.

12.7.2. El modo texto de la VGA

Así pues, en modo texto, los 4 bits de color indexan 16 posibles registros de paleta. Cada registro de paleta indexa a su vez uno de entre los 256 registros DAC, y a su vez cada uno de éstos contiene tres grupos de 6 bits con las componentes RVA.



La ROM-BIOS proporciona una gran cantidad de funciones que permiten manipular tanto los registros de paleta como los registros DAC. A continuación se da una pequeña biblioteca de funciones que nos permitirán usar en nuestros programas cualquiera de los 262144 colores posibles, bien entendido que sólo podemos visualizar 16 de ellos simultáneamente.

En modo texto, el octeto de atributos contiene el color del texto y el color del fondo. El parpadeo del carácter es un atributo aparte, y se controla mediante la activación de un bit en un registro especial del subsistema de video. Cuando este bit se pone a 1, el bit de orden más alto de cada octeto de atributo no se interpreta como parte del color de fondo del carácter, sino que indica que el carácter debe parpadear. Por tanto, sólo quedan disponibles 3 bits con los que se pueden definir 8 colores para el fondo. Existe una función del DOS que permite activar y desactivar el bit responsable del parpadeo, y que por tanto nos permite usar los 16 colores como fondo:

```
int establece_color(int indDAC, int R, int V, int A)
{
  /*
  establece componentes RVA para una entrada DAC
  */
  struct REGPACK r;
  r.r_ax=0x1010;
  r.r_dx=R;
  r.r_cx=(V<<8)+A;
  r.r_bx=indDAC;
  intr(0x10,&r);
  return(0);
}

int lee_paleta(int indDAC, int *R, int *V, int *A)
{
  /*
  lee componentes RVA para una entrada DAC
  */
  struct REGPACK r;
  r.r_ax=0x1015;
  r.r_bx=indDAC;
  intr(0x10,&r);
  *R=(r.r_dx>>8);
  *V=(r.r_cx>>8);
  *A=(r.r_cx<<8)>>8;
  return(0);
}
```

```
int indexa_color(int indcolor, int indDAC)
{
   /*
   hace que una entrada del registro de paleta apunte a una
   entrada DAC particular
   */
   struct REGPACK r;
   r.r_ax=0x1000;
   r.r_bx=(indcolor<<8)+indDAC;
   intr(0x10,&r);
   return(0);
}

int indiceDAC(int indcolor)
{
  /*
  dice a que registro DAC apunta una entrada del registro
  de paleta
  */
  struct REGPACK r;
  r.r_ax=0x1007;
  r.r_bx=indcolor;
  intr(0x10,&r);
  return(r.r_bx>>8);
}

int fondo16(int on_off)
{
  /*
  actua sobre el bit de parpadeo: 0-> inhibe; 1-> activa
  */
  struct REGPACK r;
  r.r_ax=0x1003;
  r.r_bx=on_off;
  intr(0x10,&r);
  return(0);
}
```

En realidad, el esquema sencillo mostrado anteriormente por el que se relacionan los registro de paleta con los registro DAC es algo diferente. Existen dos registros cuyo concurso es fundamental. El primero se denomina registro de control de modo (RCM) y el segundo registro de selección de color (RSC). Cuando el bit 7 del RCM se encuentra a cero, la entrada DAC a la que apunta el registro de paleta se forma con los bits cero al cinco del registro de paleta y los bits dos y tres del RSC. De esta forma, la tabla DAC se divide en cuatro grupos de sesenta y cuatro colores. Por el contrario, cuando el bit siete del RCM se encuentra a uno, la entrada DAC se forma con los bits cero a tres del registro de paleta y con los bits cero a tres del RSC, y por tanto la tabla DAC queda dividida en dieciséis grupos de dieciséis colores cada uno. Para establacer el modo, se usa la función 0x13, cuyo funcionamiento es el siguiente: cuando se invoca con el valor 0 en BL, el bit 0 de BH se copia en el bit 7 del RCM. Según el valor de este bit, la tabla DAC queda dividida en cuatro o dieciseis grupos. Por el contrario, cuando la función se llama con BL a uno BH se copia en el RSC. De esta forma podemos referirnos a una entrada determinada de la tabla DAC.

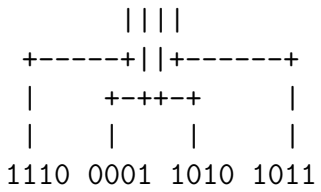
12.7.3. Modos gráficos

En los modos gráficos de la VGA, el mapa del *buffer* de video, que comienza en A000:0000H, es lineal, al igual que en los modos de texto. Los *pixels* están almacenados de izquierda a derecha en cada octeto, y una fila de *pixels* sigue a otra en el mapa de memoria.

Sin embargo, en la EGA y VGA hay que tener en cuenta que el *buffer* gráfico en los modos de 16 colores está constituido por un conjunto de cuatro mapas de memoria paralelos, ya que la memoria está configurada para tener 4 mapas de 64KB ocupando el mismo rango de direcciones que comienza en A000H:0000H.

Tanto EGA como VGA tienen una circuitería especial para acceder a estos mapas de memoria en paralelo, de forma que cada *pixel* de cuatro bits está almacenado con un bit en cada mapa de memoria, tal y como se muestra en la figura:

```
10110101...  Mapa 3
10000000...  Mapa 2
10110101...  Mapa 1
01011110...  Mapa 0
```



El motivo para esta arquitectura se comprende al examinar el modo 640x350 a 16 colores. Cada *pixel* requiere 4 bits, y se necesitan por tanto 640 x 350 x 4 = 112000 octetos, lo que supera el tamaño máximo de uno de los segmentos del 8086, que es de 64KB. Pero organizando los datos de los *pixels* en paralelo, sólo se necesitan 640 x 350 = 28000 octetos por cada mapa de memoria. Dada la variedad de mapas de memoria y modos gráficos, es una suerte que la ROM-BIOS proporcione procedimientos para leer y escribir *pixels* individuales, aunque el precio es que estos son bastante lentos.

Los distintos modos gráficos se seleccionan mediante la función 0x00 de INT 0x10: AH=0x00 y AL=modo de video, donde los modos de video posibles son:

Código	Resolución	Modo	Colores
00h	40x25	Texto	16
01h	40x25	Texto	16
02h	80x25	Texto	16
03h	80x25	Texto	16
04h	300x200	Gráficos	4
05h	300x200	Gráficos	4
06h	640x200	Gráficos	2
07h	80x25	Texto	MONO
0dh	300x200	Gráficos	16
0eh	640x200	Gráficos	16
0fh	640x350	Gráficos	MONO
10h	640x350	Gráficos	16
11h	640x480	Gráficos	2
12h	640x480	Gráficos	16
13h	320x200	Gráficos	256

La escritura de puntos gráficos mediante la BIOS es lenta, como se ha dicho. De ello se encarga la función 0xCH de INT 0x10, que escribe un punto gráfico.

```
ENTRADA:  AH = 0xch
          DX = Linea de la pantalla
          CX = Columna de la pantalla
          AL = Valor del color
```

El valor del color depende del modo gráfico actual. Por ejemplo , en el modo 640 x 200 sólo se permiten los valores 0 y 1, mientras que en 320 x 200 se permiten valores entre 0 y 3.

Análogamente, es posible leer el valor del color de un punto gráfico mediante la función 0x0D:

```
ENTRADA: AH = 0x0D
         DX = Linea
         CX = Columna
SALIDA:  AL = Valor del color
```

La BIOS de EGA y VGA suministra montones de funciones nuevas. La 0x1B por ejemplo devuelve toda la información del modo de video actual:

```
ENTRADA: AH = 0x1B
         BX = 0
         ES:DI = Puntero a un buffer
SALIDA:  AL = 0x1B
```

Si después de la llamada a esta función no se devuelve 0x1B en AL , significa que no hay VGA instalada. En ES:DI se pasa la dirección de un *buffer* de 64 octetos donde la función colocará toda la información sobre el modo de video.

En *PC Interno*, de M. Tischer, ed. Marcombo, pag. 1193, puede encontrarse el mapa de este área de memoria.

Particularmente interesante es la función 0x1C, que permite desde un programa de aplicación guardar el estado actual de video para recuperarlo al finalizar. Antes de esto, es necesario determinar el tamaño del *buffer* necesario para almacenar la información. De esto se encarga la subfunción 0x00:

```
ENTRADA: AH = 0x1C
         AL = 0
```

```
         CX = Componentes a guardar
SALIDA:  AL = 0x1C
         BX = tamanyo del buffer en unidades de 64 octetos
```

Si en AL no se devuelve 0x1C, no hay instalada VGA. Los tres bits inferiores de CX determinan qué es lo que se va a guardar:

```
Bit 0 a 1: Video_Hardware (?)
Bit 1 a 1: Zona de datos de la BIOS VGA
Bit 2 a 1: Contenido de los registros DAC
```

En cualquier caso, el contenido de la RAM de video ha de ser guardado por el programa de aplicación. Una vez obtenido el tamaño del *buffer* que necesitamos, la subfunción 0x01 se encarga de guardar la información:

```
ENTRADA: AH = 0x1C
         AL = 1
         CX = Componentes a guardar
      ES:BX = Puntero al buffer
```

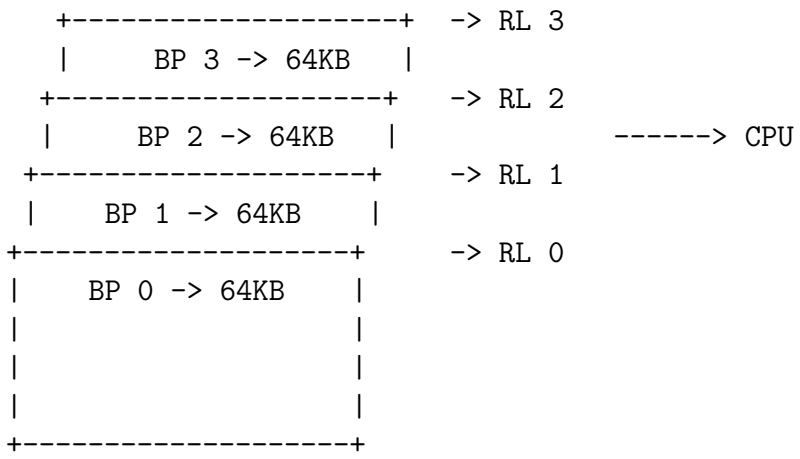
De restaurar el modo de video original se encarga la subfunción 0x02:

```
ENTRADA: AH = 0x1C
         AL = 2
         CX = Componentes a restaurar
      ES:BX = Puntero al buffer
```

12.7.4. Los *bit-planes*

El aumento de la resolución y el número de colores que se ha producido, ha incrementado la cantidad de RAM necesaria hasta el punto de desbordar completamente el espacio asignado originalmente por debajo del primer megaocteto de RAM. Surge por tanto la cuestión de como acomodar los, en principio, 256KB de las tarjetas EGA y VGA, ya que, en el espacio direccionable, sólo se cuenta con el segmento 0xA000. Para superar esta limitación se introdujeron los *bit-planes* (BP en adelante).

La CPU lee y escribe en la RAM de video a través de cuatro registros de un octeto llamados *registros latch* (RL). Estos cuatro octetos no son accesibles directamente, y cada uno se corresponde con un BP. Cada BP tiene 64KB, de modo que los cuatros BP hacen 256KB, que es la memoria base con que cuentan las tarjetas EGA/VGA



Pero durante un proceso de lectura sólo se transfiere un octeto, y por tanto surge la pregunta de cual de los RL es el que finalmente se lee. En cuanto a la escritura, ¿cuál de los RL y en que BP se escribe? La respuesta a ésta y otras preguntas similares se encuentra en los nueve registros del controlador gráfico, parte integrante de toda tarjeta EGA/VGA.

A él se accede mediante la dirección de puerto 0x3CE, donde en primer lugar se carga el número de registro que se quiere escribir, y el valor deseado se envia al puerto 0x3CF.

Presentaremos algunos de los registros comúnmente usados. Uno de los principales es el registro de modo, que es el numero 0x05. Los bits cero y uno de este registro determinan el modo de escritura, y el bit tres el modo de lectura. En todo acceso de lectura a la RAM de video se cargan los cuatro RL, pero sólo uno de ellos llega a la CPU. En el modo de lectura cero puede elegirse el BP que se desea leer, y este BP se determina por el registro 0x04, en concreto, por el valor de sus dos bits más bajos. El siguiente fragmento de código (Tischer, página 268), muestra como leer 8KB del segundo BP y colocarlos en memoria:

```
MOV AX,DS
MOV ES,AX
MOV DI,OFFSET BUFFER  ; es:di apunta al buffer
MOV AX,A000H
MOV DS,AX
XOR SI,SI             ; ds:si apunta a A000:0000H
MOV CX,8*1024
```

```
MOV DX,3CEH
MOV AX,0005H          ; valor cero en registro cinco para
                      ; seleccionar modo de lectura cero
OUT DX,AX
MOV AX,0104H          ; valor 1 en registro 4 para leer del
OUT DX,AX             ; segundo BP

REP MOVSB
```

En el segundo modo de lectura, modo 1, se efectúan diversas operaciones lógicas con los cuatro RL, con objeto de obtener el octeto que es enviado finalmente a la CPU. Con los bits correspondientes de los cuatro RL se forman ocho grupos de cuatro bits, que se comparan con el valor del registro 0x02, y del resultado de la comparación se obtiene el octeto que se envia a la CPU. Por ejemplo, dados los cuatro registros latch:

```
10101010
00111111
01001111
01110011
```

y dados los cuatro bits bajos del registro 0x02 por 0110, el octeto resultante de la lectura es (00000100).

(En efecto: para el bit bajo la comparación es ¿0111=0110?, y la respuesta es no, luego el bit bajo del octeto resultado es un cero. Para el bit tres del octeto resultante la comparación ¿0110=0110? es cierta, y por tanto vale 1, etc.).

Como ejemplo, tras la ejecución del siguiente fragmento de código, los bits a uno en AL se corresponden con los grupos de 4 bits cuyo valor es cinco:

```
MOV AX,A000H
MOV ES,AX
MOV DX,3CEH
MOV AX,0805H          ; registro cinco, modo read 1
OUT DX,AX
MOV AX,0502H          ; registro dos, valor cinco
OUT DX,AX
MOV AL,ES:0
```

De los tres modos de escritura, el modo cero es el más común. En este modo se realizan una serie de operaciones que dependen del contenido de varios registros. Los bits del registro 0x08 se corresponden con los bits de los cuatro RL, que en un acceso de escritura se cargan con el mismo valor. El que un bit del registro 0x08 esté a cero o a uno determina si el bit correspondiente del RL ha de escribirse sin cambios o si han de realizarse operaciones con él antes de la escritura. Qué tipo de operaciones han de realizarse lo determinan los bits 4 y 3 del registro 0x03, que pueden tomar los valores:

```
00 = Sustitucion
01 = AND
10 = OR
11 = XOR
```

¿Y cuál es el segundo operando? El bit correspondiente del registro 0x01. En definitiva, el bit i del registro latch se opera con el bit i del registro 0x01 si el bit i del registro 0x08 se encuentra a uno. Qué tipo de operación se decide a través del registro 0x03. En contraste con el modo de escritura cero, el modo de escritura uno es tan simple como escribir el contenido de los cuatro RL directamente en cada uno de los BP. Otra parte importante de la tarjeta EGA/VGA es el *Controler Sequencer*, que dispone al igual que el controlador gráfico de una serie de registros, cada uno con una misión específica. En concreto, el llamado *Map-Mask-Register* determina a través de sus cuatro bits bajos qué BP se encuentra o no accesible. En definitiva, se posee control absoluto sobre qué valor se escribe en cada octeto de cada BP, y sobre la procedencia de cada valor leído.

12.7.5. La especificación VESA

De la exposición anterior, debería de quedar claro que la programación de la VGA es sumamente engorrosa, y que requiere un considerable esfuerzo el obtener el rendimiento que la tarjeta es capaz de ofrecer. Por este motivo, los programadores siempre se han decantado por los modos de video lineales, y durante mucho tiempo la mayoría de los juegos se realizaron en el modo 320x200 con 256 colores, donde un segmento puede acomodar toda la información de la pantalla. Resoluciones superiores requieren la conmutación de bancos, un proceso lento. Además, la obtención de prestaciones adicionales es dependiente del fabricante. Esta situación dio lugar a la aparición de una norma, apoyada por algunos de los más influyentes fabricantes. Esta norma, denominada VESA, especifica procedimientos para que un programa pueda

interrogar a la tarjeta sobre los modos de video de que dispone y mecanismos también comunes para activar estos modos y usarlos.

La función 0x4f de la BIOS, subfunción 0x00 de la interrupción 0x10, obtiene las características de la tarjeta VESA. Esta función toma como entrada 0x4f00 en AX y la dirección del *buffer* donde la función depositará la información sobre la tarjeta en ES:DI. A la salida, si `AX=0x004f` la tarjeta soporta VESA. De lo contrario, no se soporta VESA. Este es el código de la función:

```
int VESA(unsigned char *buffer)
{
    struct REGPACK reg;
    reg.r_ax=0x4f00;
    reg.r_es=FP_SEG(buffer);
    reg.r_di=FP_OFF(buffer);
    intr(0x10,&reg);
    if ((reg.r_ax & 0x00ff)==0x004f) return(1);
    else return(0);
}
```

Suponiendo que la llamada haya tenido éxito, la información se encontrará depositada en el *buffer* , de 256 octetos al menos, y estructurado de la siguiente forma:

```
desplaz.   Contenido
========   =========
0x00     cadena "VESA"
0x04     version VESA (numero mayor)
0x05     version VESA (numero menor)
0x06     puntero a la cadena con el nombre del fabricante
0x0a     de momento no se usa
0x0e     puntero a la lista con los numeros de codigo
         de los modos soportados
```

Los números de código son palabras de 16 bits, y la lista termina con el código 0xffff. Por ejemplo, para obtener el nombre del fabricante y los modos soportados, el código necesario es el siguiente (insisto: siempre que la llamada a la función anterior haya tenido éxito):

```
unsigned long *a;       /* para apuntar a los punteros  */
unsigned int *q;        /* para recorrer lista de modos */
```

```
unsigned char *pc;      /* puntero al buffer informativo*/
unsigned char *aux;     /* auxiliar para recorrer buffer*/
int n;

pc=(unsigned char *)malloc(256);

if (VESA(pc)){
    printf("BIOS VESA disponible! | ");
    aux=pc;
    printf("signatura: ");
    for(n=0;n<4;++n,++aux) printf("%c",*aux); printf(" | ");
    printf("version VESA: %d.%d\n",*(aux+1),*(aux+2));
    a=pc+6;
    printf("Fabricante tarjeta: %s\n",*a);
    a=pc+14;
    q=(*a);

    /* imprimir la lista de modos */
    printf("Modos disponibles");
    while ((*q)!=0xffff){
        printf(" %3xh -> ",*q);
        switch (*q){
        case 0x100: printf("  640*480    16c   256kB"); break;
        case 0x101: printf("  640*480   256c   512kB"); break;
        case 0x102: printf("  800*600    16c   256kB"); break;
        case 0x103: printf("  800*600   256c   512kB"); break;
        case 0x104: printf(" 1024*768    16c   512kB"); break;
        case 0x105: printf(" 1024*768   256c  1024kB"); break;
        case 0x106: printf("1280*1024    16c  1024kB"); break;
        case 0x107: printf("1280*1024   256c  1280kB"); break;
        default   : printf("  ?");
    }
    ++q;
}
```

Una vez obtenida la lista de modos soportados, es preciso obtener la información necesaria sobre un modo en particular. De esto se ocupa la subfunción 0x01 de la función 0x4f de la interrupción 0x10. Se invoca poniendo AX=0x4f01, en CX el código del modo VESA elegido y en ES:DI un puntero a un *buffer* donde la función depositará la información sobre el modo en

particular. A la salida, AX=004f si la llamada se ejecutó con éxito.

```
int infoVESA(int modo, unsigned char *buffer)
{
    struct REGPACK reg;
    reg.r_ax=0x4f01;
    reg.r_cx=modo;
    reg.r_es=FP_SEG(buffer);
    reg.r_di=FP_OFF(buffer);
    intr(0x10,&reg);
    if ((reg.r_ax & 0x00ff)==0x004f) return(1);
    else return(0);
}
```

Respecto a la estructura del *buffer* en cuestión, es la siguiente:

```
desplaz.   Contenido
========   =========
0x00       bandera de modo
0x02       banderas para la primera ventana
0x03       banderas para la segunda ventana de acceso
0x04       granularidad en KB
0x06       tama#o de las dos ventanas de acceso
0x08       segmento de la primera ventana
0x0a       segmento de la segunda ventana
0x0c       puntero a la rutina para ajustar la zona
           visible de las dos ventanas
0x10       numero de octetos que ocupa cada linea
```

A partir del desplazamiento 0x12 y hasta el 0x1c se encuentra información opcional, que puede estar o no según lo indiquen las banderas de modo. (ver Tischer). En cuanto a la bandera de modo en sí, codifica las siguientes informaciones, donde 0 indica 'no' y 1 indica 'si':

```
bit   significado
===   ===========
0     se puede conmutar a ese modo con el monitor enchufado ?
1     se devolvieron las informaciones opcionales ?
2     se soportan las funciones de texto de la BIOS en ese modo ?
3     es un modo de color ?
```

```
4     es un modo grafico ?
5-7   no usados
```

Por su parte, las banderas de ventana codifican las siguientes informaciones:

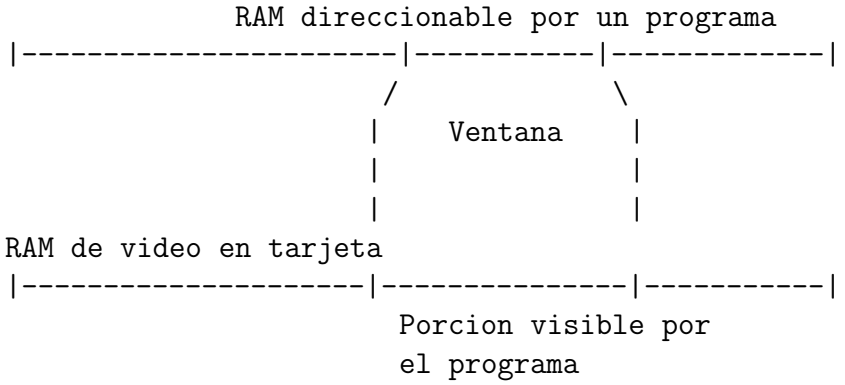
```
Tabla 3
-------
bit   significado
===   ===========
0     Esta la ventana disponible ?
1     son posibles accesos de lectura a traves de esa ventana ?
2     son posibles accesos de escritura a traves de esa ventana ?
3-7   no usados
```

Una vez realizado todo este trabajo preliminar, estamos en disposición de conectar un modo determinado, si hemos deducido que ello es posible. Para eso usaremos la subfunción 0x02, colocando en BX el código del modo deseado. Como siempre, a la salida AX=0x004f si la función se ejecutó con éxito:

```
int setVESA(int modo)
{
    struct REGPACK reg;
    reg.r_ax=0x4f02;
    reg.r_bx=modo;
    intr(0x10,&reg);
    if ((reg.r_ax & 0x00ff)==0x004f)
        return(1);
    else return(0);
}
```

El siguiente paso consiste en hacer corresponder una parte de la RAM de video de la tarjeta con la ventana que tenemos disponible, a través de uno de los segmentos devueltos en la Tabla 1, y por tanto hacerla direccionable por un programa. De eso se ocupa la subfunción 0x05/0x00, que toma como entradas AX=0x4f05, BH=0x00, BL=ventana de acceso (0 o 1) DX=dirección de acceso. La dirección de acceso de la RAM de video que se hará corresponder con la memoria direccionable por un programa, se dará en unidades de granularidad, que se obtuvo mediante una llamada previa a la función 0x01. Como siempre, la salida es AX=0x004f si la llamada se ejecutó correctamente:

```
int setVentana(int ventana, int RAM_tarjeta)
{
    struct REGPACK reg;
    reg.r_ax=0x4f05;
    reg.r_dx=RAM_tarjeta;
    reg.r_bx=ventana;
    intr(0x10,&reg);
    if ((reg.r_ax & 0x00ff)==0x004f)
        return(1);
    else return(0);
}
```



La subfunción 0x0501 es la complementaria de la anterior, y permite obtener la posición en la RAM de la tarjeta de la ventana que esté usando el programa:

```
int getVentana(int ventana)
{
    struct REGPACK reg;
    reg.r_ax=0x4f05;
    reg.r_bx=256+ventana;
    intr(0x10,&reg);
    if ((reg.r_ax & 0x00ff)==0x004f)
        return(reg.r_dx);
    else
        return(-1);
}
```

Habiendo pués recorrido este largo camino, estamos en disposición de escribir un *pixel* en pantalla. Supongamos que tenemos el modo de video 800x600 con 256 colores. Necesitamos en total 480,000 octetos, es decir, 7 segmentos completos y una fracción de un octavo, por tanto, 8 segmentos. Un *pixel* en la fila 'f' columna 'c' de la pantalla ocupa la posición 800*f+c (tomando el origen en (0,0)). Por tanto, la dirección líneal de un *pixel* de coordenadas (f,c) sobre el arreglo de 480,000 octetos que representan la pantalla en este modo en particular es l=800*f+c. Este octeto pertenece al segmento $s = (l \gg 16)$. Si la granularidad es de 16KB, pongamos por caso, el primer segmento comienza en la dirección 0 en unidades de granularidad, el segundo en la dirección 4, el tercero en la dirección 8, etc. En general, si la granularidad es n KB, el segmento s comienza en la dirección $d = 64 * s/n$. Entonces, debemos mover la ventana a la posición d mediante la función que hemos llamado anteriormente `setVentana`, acceder al desplazamiento del *pixel* en la ventana, que es $l - (s \ll 16)$, y escribir directamente. Por ejemplo, supongamos que deseamos escribir un *pixel* en la fila 312 columna 80. Su dirección líneal es 312*800+80=249,680. Esta posición se encuentra en el cuarto segmento (numero 3), con un desplazamiento de 53,072. Si la granularidad fuese de 16KB, la dirección del cuarto segmento seria 16.

Para terminar, digamos que la especificación VESA 2.0 admite *buffer* lineal en la memoria por encima del primer MB, pero para usar esta capacidad es preciso disponer de un compilador de 32 bits. Uno de los más populares es la versión para DOS del compilador C de GNU. En esencia, sólo es necesario reservar un vector de tanta memoria de video como sea precisa para almacenar una pantalla completa y pasar la dirección del vector a la tarjeta. Se procede entonces escribiendo directamente en el arreglo. Los detalles sobre estas capacidades de 'djgpp' pueden encontrarse en las páginas info, en las FAQ o en webs especializadas en programación de juegos.

12.8. Algoritmos gráficos

Aunque el estudio de los variados algoritmos gráficos cae fuera de la competencia de estos apuntes, es evidente que merecen una mención especial, dada la relación de dependencia de la representación gráfica con el uso de algoritmos adecuados. Se dirá que la necesidad de algoritmos adecuados surge en cualquier rama de la programación, cierto, pero es en la programación gráfica donde su ausencia se hace más evidente. Un usuario quizás no note la elección por parte del programador del algoritmo idóneo en un problema de cálculo: quizás esta elección solo se manifieste en unos pocos segundos adi-

cionales de espera. Pero en el caso de la programación gráfica los resultados son visibles directamente.

Por su uso intensivo, revisten una especial importancia los algoritmos para el trazado de líneas rectas. El trazado de líneas paralelas a los ejes coordenados de la pantalla, y formando un ángulo de 45 grados con los mismos, no reviste problemas. Pero a distintos ángulos surge el problema de que la línea pasa por puntos que en realidad no existen sobre la pantalla (recordemos que esta es una matriz discreta). La elección de qué puntos sobre la pantalla sustituyan a los puntos de la recta tiene gran importancia. Independientemente de esto, la línea tiene que comenzar y terminar correctamente, y, lo que es más difícil de conseguir cuando se usan puntos que no pertenecen a la recta para aproximarla, el trazo aproximado ha de tener densidad constante en todo su recorrido. Este requerimiento, por último, ha de cumplirse cualquiera que sea el ángulo elegido, y el algoritmo que se use ha de ser rápido.

Uno de los mejores algoritmos es el debido a Bresenham. Está diseñado de forma que en cada iteración una de las coordenadas x o y del siguiente punto perteneciente a la trayectoria aproximada cambia en una unidad respecto al punto actual. La otra coordenada puede o no cambiar dependiendo del valor de una variable de error mantenida por el programa y que mide la distancia, perpendicular al eje que forma un ángulo menor con la dirección del movimiento, entre la trayectoria ideal y los puntos de la trayectoria aproximada. Supongamos en la descripción siguiente que el eje que forma un angulo menor con la línea es el eje x. En cada iteración del algoritmo, la pendiente de la recta se suma al error e, y el signo del mismo es usado para decidir si el siguiente punto a dibujar tendrá incrementada la coordenada y o no. Un valor positivo indica que la recta real pasa por encima del punto actual, y por tanto se incrementará la coordenada y del siguiente punto y se reducirá en 1 el valor de e. Un valor negativo deja invariante y. El algoritmo se dá a continuación en PASCAL:

```
procedure bresemh1;
var
x,y,deltax,deltay:integer;
e:real;
begin
    e:=(deltay/deltax)-0.5;
    for i:=1 to deltax do begin
    plot(x,y);
    if e>0 then begin
```

```
        inc(y);
        dec(e);
    end;
    inc(x);
    e:=e+(deltay/deltax);
    end;
end;
```

El punto débil del anterior algoritmo se encuentra en la primera de las divisiones que hay que realizar, pero, como en realidad no nos interesa el valor de `e` sino sólo su signo, el algoritmo queda invariante si multiplicamos `e` por una constante, que puede ser `2*deltax`. Entonces podemos escribir:

```
procedure bresemh2;
var
x,y,deltax,deltay:integer;
e:real;
begin
    e:=2*deltay-deltax;
    for i:=1 to deltax do begin
    plot(x,y);
    if e>0 then begin
        inc(y);
        e:=2*deltay-2*deltax;
    end else
        e:=e+2*deltay;
    inc(x);
    end;
end;
```

Cuando `deltay >deltax` el algoritmo es similar, intercambiando los papeles de `x` e `y`. Este algoritmo que evita multiplicaciones y divisiones, puede mejorarse aún más para funcionar con aritmética entera y evitar la generación de puntos duplicados.

Otro de los elementos gráficos del que se hace uso intensivo es el arco de circunferencia. Limitaremos la discusión a circunferencias centradas en el origen, ya que cualquier otra circunferencia se puede obtener por simple traslación. La ecuación diferencial de la circunferencia centrada en el origen es:

$$\frac{dy}{dx} = -\frac{x}{y}. \tag{12.6}$$

Si llamamos $x(k)$ e $y(k)$ a los puntos k-*ésimos* de la sucesión que dibuja la circunferencia, dado un punto $x(n), y(n)$ cuyo vector de posición forma un angulo t con el eje horizontal, el punto $x(n+1), y(n+1)$ se determina a partir de las ecuaciones:

$$x(n+1) = x(n)cos(t) + y(n)sen(t) \tag{12.7}$$

$$y(n+1) = y(n)cos(t) - x(n)sen(t) \tag{12.8}$$

El cálculo de los senos y cosenos es lento, salvo que se encuentren precalculados, de manera que suele usarse el algoritmo siguiente:

$$x(n+1) = x(n) + ey(n) \tag{12.9}$$

$$y(n+1) = y(n) - ex(n+1) \tag{12.10}$$

donde e es 2^{-n}, y n se elige de tal forma que, si r es el radio del círculo, se cumpla $2^{n-1} \leq r < 2^n$.

Existen métodos potentes y generales para generar cualquier tipo de curva o superficie, pero caen fuera del alcance de estas notas, de modo que remitimos al lector a la bibliografía especializada.

El otro aspecto importante a considerar, que omitimos aquí, es el de las transformaciones líneales en el plano y el espacio que permite situar, trasladar o rotar (o cualquier combinación de los elementos anteriores) los cuerpos definidos matemáticamente, así como las relaciones entre las coordenadas de los puntos pertenecientes a un cuerpo y las coordenadas de su representación en pantalla.

En definitiva, la importancia y extensión del tema, dada la actual preeminencia de la programación gráfica, aconsejarían estudiar estas cuestiones con detenimiento.