CAPÍTULO V: ENTRADA/SALIDA

ÍNDICE DE CONTENIDO

1. Los problemas de la E/S

La gestión de las entradas/salidas (E/S) es uno de los aspectos más importantes y dificultosos en el diseño de los sistemas operativos debido a la gran cantidad y variedad de periféricos.

Para disminuir la complejidad uno de los objetivos fundamentales que se plantean los diseñadores de los sistemas operativos es proporcionar una interfaz sencilla y fácil de utilizar entre los dispositivos y el resto del sistema. Lo que buscan es lograr una independencia de dispositivo esto es, lograr una interfaz que permita tratar de la misma forma a todos los dispositivos.

Este tratamiento se traduce, básicamente, en:

- Que todas las operaciones de E/S usan las mismas llamadas al sistema.

- Que se trabaja con dispositivos virtuales y no directamente con el dispositivo. Por ejemplo, es muy común trabajar con los dispositivos a través de descriptores de ficheros.

Para lograr esto, y partiendo del conocimiento profundo de los principios de funcionamiento del hardware de entrada/salida, se intentan generalizar las características y funciones de estos dispositivos y así conseguir diversas clasificaciones que permitan tratar de forma uniforme al mayor conjunto de ellos. Una de estas clasificaciones propone la siguiente división:

- **Dispositivos de bloques**: permiten leer o escribir la información aleatoria en bloques de tamaño fijo, cada uno con una dirección, con independencia de los demás (p. ej. discos duros). En estos dispositivos la unidad mínima direccionable es el **segmento** o **bloque del dispositivo**, por lo general su tamaño es de 512 bytes. Desde el punto de vista del software la mínima unidad de transferencia no es el sector sino el **bloque** o **bloques de E/S**. El tamaño de un bloque está limitado por el tamaño de un sector y el tamaño de una página y siempre tiene que ser un múltiplo del tamaño de un sector.

- **Dispositivos de caracteres**: los dispositivos de este tipo suministran un conjunto de caracteres sin estructurarlos en bloques, el acceso a la información

se realiza suministrando un flujo de caracteres por parte del dispositivo (p. ej. impresoras y los ratones).

El principal problema que presenta esta primera clasificación es que dispositivos como los relojes no encuentran cabida. Otra clasificación todavía más general es aquella que habla de:

- **Dispositivos legibles por los humanos**: aquí se incluyen todos los periféricos tanto de entrada como de salida.
- **Dispositivos legibles por la máquina**: suelen ser los dispositivos de almacenamiento ya sean volátiles o no.
- **Dispositivos de comunicaciones**: permiten conectar computadoras entre sí.

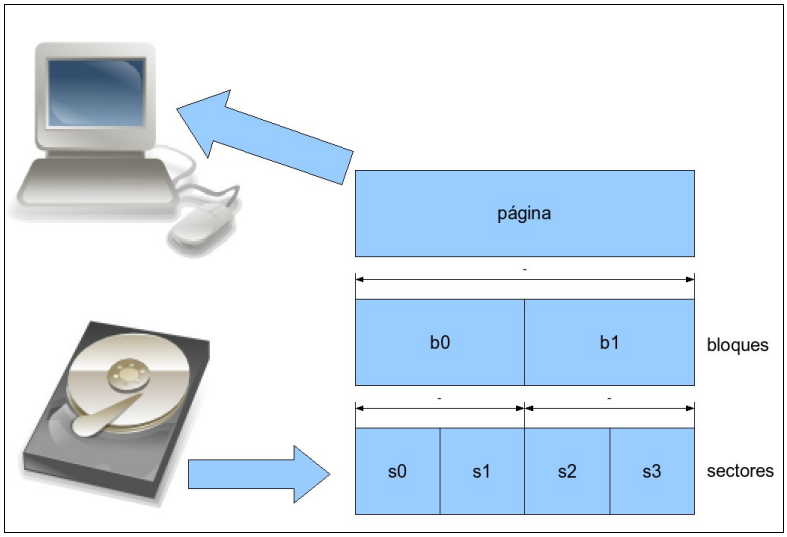


Figure V-1: Sectores, bloques y páginas

Aún con está clasificación nos podemos encontrar diferencias muy importantes entre dispositivos incluidos dentro de la misma categoría debido a características tan variantes como:

- Velocidades de los datos.
- Aplicaciones, no es lo mismo el uso de un disco para una aplicación de base de datos que para uso como dispositivo de intercambio.
- Complejidad de control.
- Unidad de transferencia, en caracteres o en bloques.
- Tiempos de respuesta (Operaciones asíncronas), no conocemos el tiempo de respuesta, con lo que debemos esperar a que finalice el dispositivo (interrupciones).
- Condiciones de error.

Ejemplo: El Linux hay tres clases de dispositivos:

- **Dispositivos de caracteres**, suelen implementar las llamadas al sistema open, close, read y write. La consola de texto (`/dev/console`) y los puertos serie (`/dev/ttyS0`) son ejemplos de este tipo de dispositivos. El acceso a estos dispositivos se realiza mediante el sistema de ficheros, como si de un fichero regular se tratara.

- **Dispositivos de bloques**. Linux permite tratar a estos dispositivos como los anteriores. Aún así son más complejos de gestionar que los de caracteres y el interfaz entre el kernel y el driver es diferente.

- **Interfaz de red**. Son dispositivos que permiten enviar datos entre dos computadores. El acceso a estos dispositivos no se realiza mediante el sistema de ficheros sino mediante un nombre único (como eth0).

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

Figure V-2: Diferencias de velocidad entre dispositivos

Así pues, podemos resumir todo lo comentado hasta ahora indicando que el software del sistema operativo dedicado a manejar los dispositivos de entrada/salida debería proporcionar:

- Órdenes para facilitar el uso del periférico.
- Aceptación y tratamiento de interrupciones.
- Gestión de errores.

2. Principios del hardware de entrada/salida

Como hemos dicho antes, es importante tener claros ciertos aspectos generales del hardware de E/S y su relación con la programación antes de ponernos a crear un software que los controle. En los siguientes puntos vamos a hablar de conceptos tan básico como:

- Conexión entre periféricos y CPU: controladoras y canales.
- Comunicación CPU y controladoras: puertos y E/S con correspondencia en memoria.
- Control de la E/S: E/S programada, E/S mediante interrupciones y acceso directo a memoria.

2.1. Conexión entre dispositivos de E/S y la CPU

La comunicación entre el procesador y los dispositivos de entrada/salida se realiza a través de unas conexiones eléctricas denominadas líneas. Estas líneas se agrupan, teniendo en cuenta sus funciones, en buses de entre los que podemos destacar los de direcciones, control y datos.

Inicialmente el procesador controlaba directamente los dispositivos de E/S pero las diferencias de velocidad entre la CPU (Ghz), la RAM (nanosegundo) y los dispositivos de entrada/salida (milisegundos o más), convertían a las operaciones de entrada/salida en verdaderos cuellos de botella.

Para solucionar este problema se aisla al procesador de los detalles específicos del dispositivo, se realiza una conexión indirecta entre ambos a través de:

- **Controladoras**.
- **Canales**.

Controladores

Las unidades de E/S suelen constar de un componente mecánico (el propio dispositivo) y uno electrónico (**controladora o adaptador de dispositivo**).

La CPU se conecta a través de esta controladora al dispositivo:

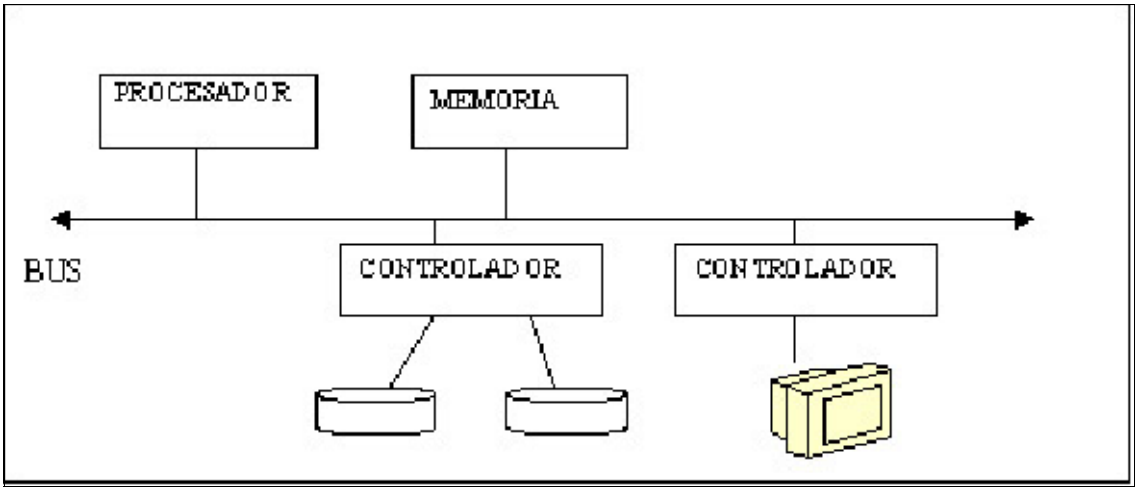


Figure V-3: Controladoras

Una controladora puede servir para varios dispositivos del mismo tipo. Por ejemplo, las controladoras de disco ATA pueden controlar hasta 4 dispositivos del mismo tipo.

La interfaz entre la controladora y el dispositivo es estándar como: IDE (Integrated Device Electronics), SCSI (Small Computer Systems Interface), USB (Universal Serial Bus), etc.

Figure V-4: Controladora SCSI

La tareas de la controladora consisten en:

- Almacenar el estado dispositivo.
- Controlar el dispositivo.
- Convertir el flujo de bits en una serie de bloques de bytes y realizar la corrección de errores necesaria.

Canales

Otra forma de conexión es a través de canales de E/S, que consiste en un procesador específico para operaciones de E/S.

La evolución de estos procesadores ha llevado a que tengan su propia memoria, se habla ahora de **procesadores de E/S** (p. ej. familia IOP de Intel). El canal pretende tratar al dispositivo como abstracto o virtual.

Se maneja mediante ordenes especializadas. Una vez terminada la acción el canal devuelve el estado del dispositivo a la vez que interrumpe al procesador central.

Al ser estos procesadores de E/S muy simples y sin grandes exigencias de tiempo resultan muy económicos.

Los canales pueden ser:

- **Selectores**: permiten manejar varios dispositivos, pero sólo uno en cada momento. Generalmente es usado con dispositivos rápidos.

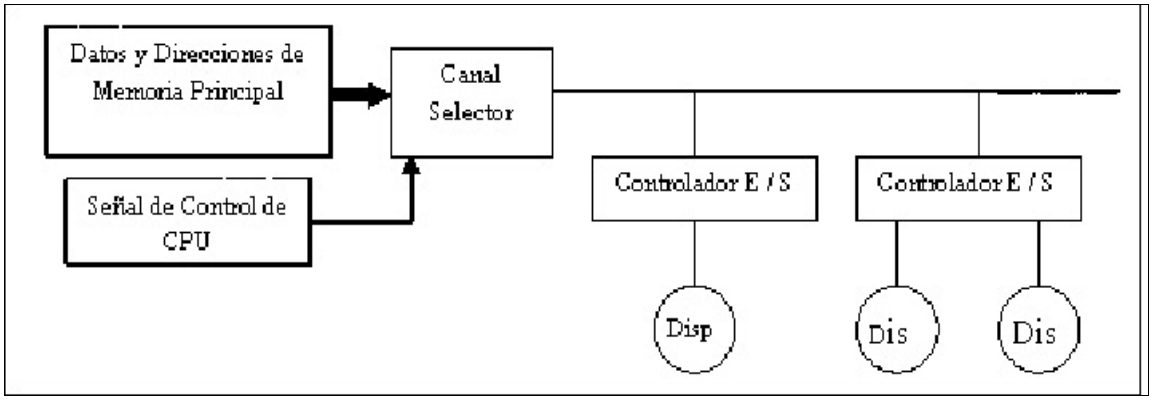


Figure V-5: Canales selectores

- **Multiplexores**: trabajan con varios dispositivos pudiendo transferir datos de todos simultáneamente. Está relacionado con dispositivos lentos como el teclado o la impresora.

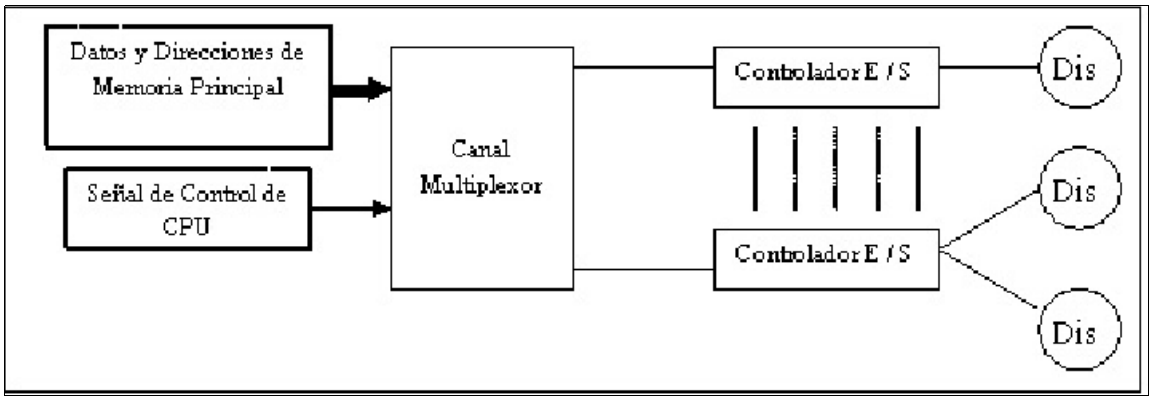
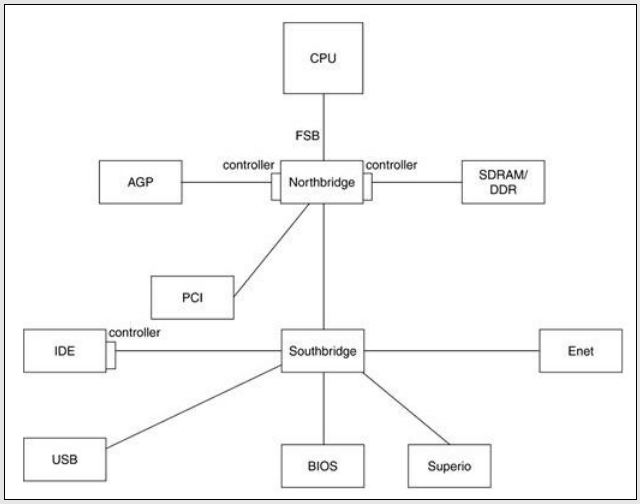


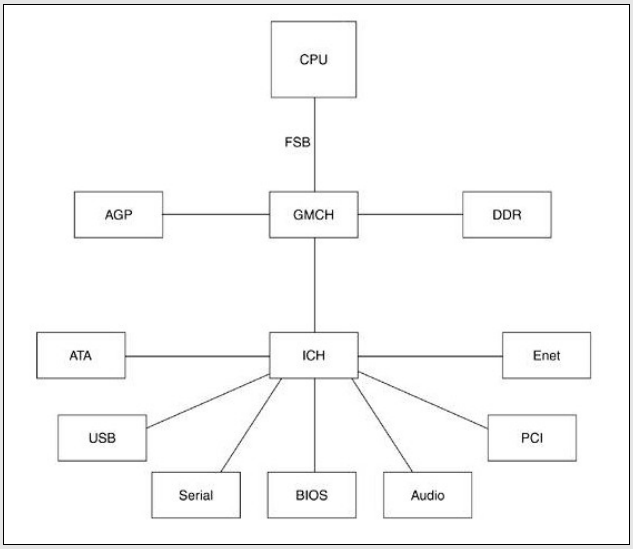
Figure V-6: Canales multiplexores

- **Multiplexor de bloques**: un canal multiplexor con canales selectores asociados.

Ejemplo: La forma de conectar los diferentes buses ha variado. En las primeras arquitecturas de Intel se usaban dos puentes para agrupar a los dispositivos de alta velocidad (puente norte o Northbridge) y los de menor velocidad (puente sur o Southbridge):



Actualmente ya no se habla de puentes sino de concentradores. El puente norte es ahora conocido como GMCH o *Graphics and Memory Controller Hub* que soporta controladores de gran velocidad como AGP o DDR. Con la aparición del PCI Express los controladores para gráficos y DDR2 han pasado a formar parte del *Memory Controller Hub* (MCH). Por otro lado al puente sur se le conoce como *I/O Controller Hub* (ICH). Ambos concentradores están conectados a través de un bus punto a punto denominado *Intel Hub Architecture* (IHA). Podemos encontrar más información sobre los chipset 865G en http://www.intel.com/design/chipsets/datashts/25251405.pdf y sobre el 925XE en http://www.intel.com/design/chipsets/datashts/30146403.pdf.



2.2. Comunicación entre la CPU y las controladoras

La CPU se comunica con las controladoras mediante registros (**registros de dispositivo** o **registros de E/S**);

Estos registros se dividen en:

- **Estado**: indica el estado en el que se encuentra el dispositivo.
- **Operación**: indica si se quiere leer, escribir, etc.
- **Datos**: es el registro donde se deposita la información necesaria para llevar a cabo la operación.

Para hacer referencia a estos registros podemos usar por cualquiera de estas dos técnicas:

- **Puertos de E/S**.
- **E/S con correspondencia en memoria**.

Puertos de entrada/salida

En este primer esquema de direccionamiento a cada registro se le asocia un número de 8 o 16 bits denominado **puerto de E/S**. El acceso al registro se realiza a través del puerto que tenga asignado:

```
//para leer de un puerto

IN REG, PUERTO

// para escribir en un puerto

OUT PUERTO, REG
```

Como se puede apreciar al aparecer el concepto de puesto es necesario añadir nuevas instrucciones (en este caso `IN` y `OUT`) que permitan acceder al nuevos espacios de direcciones donde se encuentran mapeados estos. En definitiva, tenemos dos espacios de direcciones diferentes uno para la memoria y otra para los dispositivos de entrada/salida:

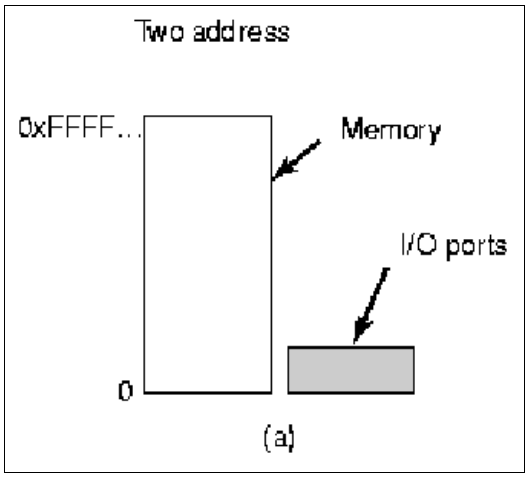


Figure V-7: Espacio de direcciones al usar puertos

Las arquitecturas Intel hacen uso habitual de esta técnica para direccionar los registros.

Entrada/salida con correspondencia en memoria

A diferencia del caso anterior se establece una relación entre los registros y el espacio en memoria, hay pues un único espacio de direcciones común:

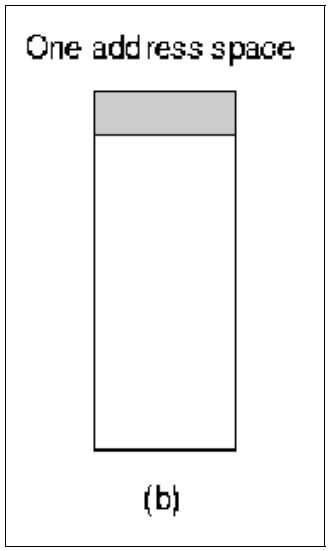


Figure V-8: Páginas de E/S

A la zona de memoria reservada para direccionar a los registros de las controladores se le denomina **página de E/S** y al conjunto de registros pertenecientes a una misma controladora y que se encuentran en esta zona se les denomina **direcciones de E/S** del dispositivo.

Las arquitecturas hacen uso de este tipo de direccionamiento ya que lo consideran más **beneficioso por** las siguientes razones:

- **No se requieren instrucciones especiales**.
- **No se necesitan nuevos esquemas de protección**.
- **Cualquier instrucción que acceda a memoria puede acceder a un registros de dispositivo**. Esta ventaja se puede convertir en un inconveniente si no tenemos en cuenta que:
  - Los registros representan dispositivos activos.
  - Los registros pueden tener restricciones de tiempo.
  - El valor que se escriba en cualquiera de estos registros no tiene por qué ser el que realmente sea objeto de operación.
  - Los registros pueden ser muy sensibles al tipo de acceso que se realice sobre los dispositivos.

Cuando Apple empezó con el diseño de sus primeras arquitecturas tuvo que solventar algunos problemas que venían dados por el uso de un único espacio de direcciones

tanto para direccionar los dispositivos de E/S como el resto de recursos del sistema. Estos **problemas** fuera:

- El **uso de la caché**, en una jerarquía de memorias clásica es fácil solventar el problema de consistencia de las caches ya que todo pasa por nuestro control. En cambio si guardamos en cache el contenido de registros de dispositivos de E/S, sobre los que no tenemos control absoluto, dichos problemas se pueden disparar. La solución tomada consiste en la desactivación selectiva de la caché.

- Sólo **hay un espacio de direcciones pero hay dos espacios de direcciones bien diferenciados**: el de memoria y el de entrada salida. Esto hace necesario que sea necesario analizar cada instrucción para ver a cuál de las dos zonas hace referencia:

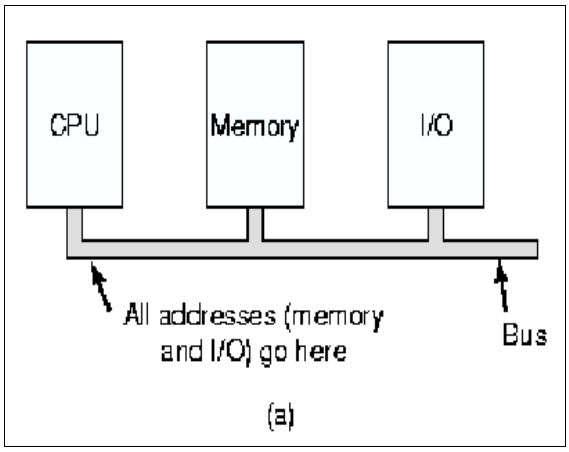


Figure V-9: Páginas de E/S

Este problema tiene una solución bastante sencilla, tener dos buses:

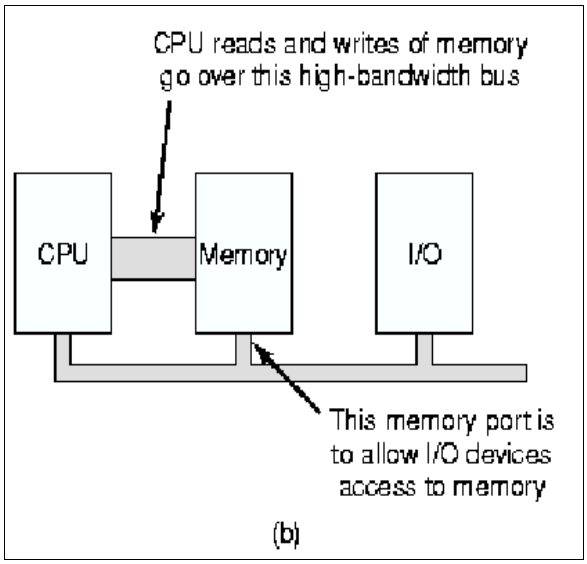


Figure V-10: Páginas de E/S

Ahora bien, cuando nos llega una dirección ¿Cómo sabe la CPU en cual de los dos buses enviar la petición? Se puede optar por:

- La envía primero a memoria y, sino hay respuesta, a los dispositivos de E/S.

- Situar un “espía” en el bus de memoria que comunica las direcciones a los dispositivos que pudieran estar interesados.
- Usar rangos de direcciones diferentes para memoria y dispositivos.

2.3. Control de entrada/salida

Otro problema es cómo conoce el hardware y el SO que una operación de E/S ha acabado:

- **E/S programada**.
- **E/S con interrupciones**.
- **E/S con DMA**.

Entrada/salida programada

Se deja que la CPU realice todo el trabajo. Supongamos que un usuario desea imprimir la cadena “ABCDEFGH”. Gráficamente la situación sería la siguiente:

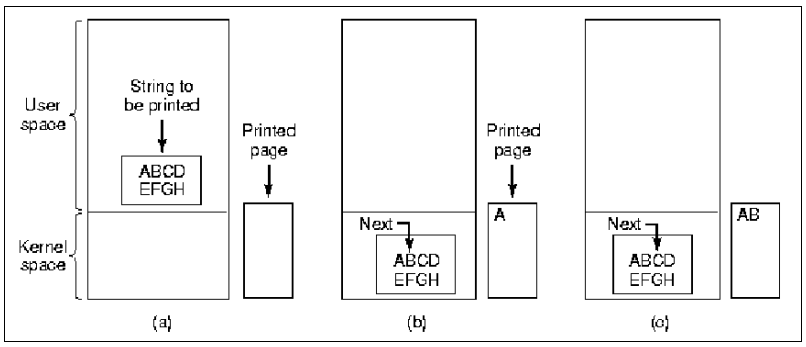


Figure V-11: Páginas de E/S

Desde el punto de vista del sistema operativo este realizaría las siguientes acciones:

```
(1)copiar_de_usuario(buffer_usr, buffer_kernel, cuenta);
(2)for (i=0; i < cuenta; i++)
(3){
(4)  while (*registro_estado_impresora != READY);
(5)    *registro_datos_impresora = buffer_kernel[i];
(6)}
(7)volver_al_usuario();
```

Observe que en (4) la CPU está interrogando al dispositivo a la espera de que este listo. Este comportamiento se conoce como **sondeo (*polling*)** o **espera activa**.

La principal ventaja que presenta este método de controlar las operaciones de entrada/salida en la sencillez. Como contrapartida la CPU está ocupada hasta que acaba la operación de E/S.

En las primeras versiones del estándar IDE se usaban los modos PIO (Programable Input/Output) para realizar las transferencias entre el disco y memoria.

Entrada/salida por interrupciones

Esta nueva manera de controlar las entrada/salida se basa en un mecanismo asistido por hardware para sincronizar el procesador con los sucesos asíncronos.

Una interrupción (o petición IRQ, Interrupt Request) es una señal que se origina en un dispositivo hardware para indicar al procesador que algo requiera su asistencia. El proceso que se sigue cuando se solicita y se resuelve una interrupción es:

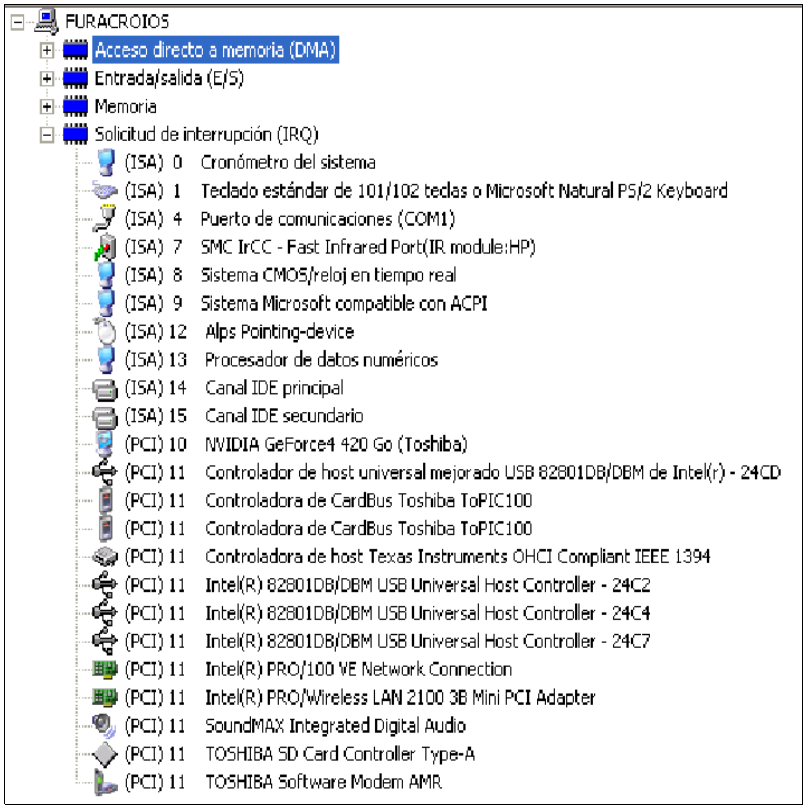


Figure V-12: Lineas de interrupción en windows

1. El dispositivo usa su línea de IRQ del bus de control para notificar su petición.

2. Un dispositivo especial denominado PIC (*Programmable Interrupt Controller*) la procesa y le indica una señal a la CPU.

3. La CPU manda una señal al PIC preguntándole por el número de interrupción.

4. El PIC le envía el número de interrupción (un número entre 0 y 256).

5. La CPU consulta la tabla de vectores de interrupción (IDT) que es una tabla indexada por número de interrupción que se encuentra en las posiciones más bajas de memoria. Contiene la dirección de memoria donde se encuentra el código a ejecutar (a este código se le denomina servicio de la interrupción -ISR- o gestor de interrupción).

6. El número de línea establece una prioridad de tal manera que las líneas con menor número (temporizador, teclado, etc.) son atendidas antes que las superiores. En caso de igualdad se ejecutan por orden.

El funcionamiento que hemos visto es válido para las interrupciones hardware. En las interrupciones software (que aparecen en las arquitecturas Intel mediante la instrucción INT) el PIC no entra en juego.

Las interrupciones pueden ser tanto enmascarables (interrumpibles) y no enmascarables. Gracias a esto podemos establecer prioridades entre ellas siendo las interrupciones hardware las más prioritarias.

Si retomamos el ejemplo de escritura en la impresora, las operaciones que ahora realizaría el sistema operativo son:

```
// código que se ejecuta cuando se realiza la
// operación de E/S
copiar_de_usuario(buffer_usr, buffer_kernel, cuenta);
habilitar_interrupciones();

while (*registro_estado_impresora != READY);

*registro_datos_impresora = buffer_kernel[0]
planifica_proceso();

// Procedimiento de servicio de interrupción
if (cuenta == 0)
{
    desbloquear_usuario
}
else
{
    *registro_datos_impresora = buffer_kernel[i];
    cuenta = cuenta -1;
    i=i+1
}
acusar_interrup();
volver_de_interrup();
```

Vemos ahora que cuando la impresora ha impreso el carácter y está preparada para aceptar el siguiente, genera una interrupción. Si bien ahora la CPU no tiene que estar pendiente de la operación de entrada/salida no es más cierto que al generarse una interrupción por carácter el rendimiento del sistema no es el mejor.

Entrada/salida por DMA

Esta técnica reduce el número de interrupciones que le llegan a la CPU. Su utilidad se ve más que justificada para dispositivos rápidos y masivos (discos) en donde es necesario que el procesador sea liberado de la realización de la transferencia de datos entre la memoria y los dispositivos. Para realizar esta tarea se usa un dispositivo denominado DMA que:

- Tiene acceso al bus del sistema independientemente de la CPU.
- Contiene varios registros que la CPU puede leer y escribir
    - Registro de dirección de memoria.
    - Registros de conteo de bytes.
    - Registros de control (puerto de la E/S, operación, unidad de transferencia, bytes leídos por ráfaga).

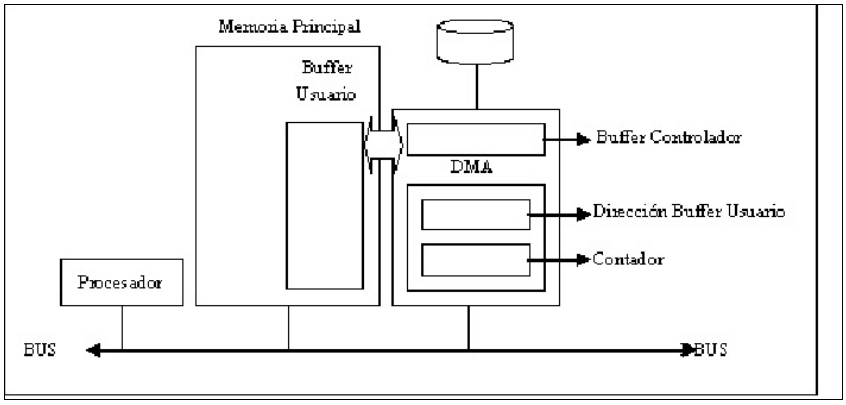


Figure V-13: Esquema general DMA

Su funcionamiento se muestra en la siguiente figura:

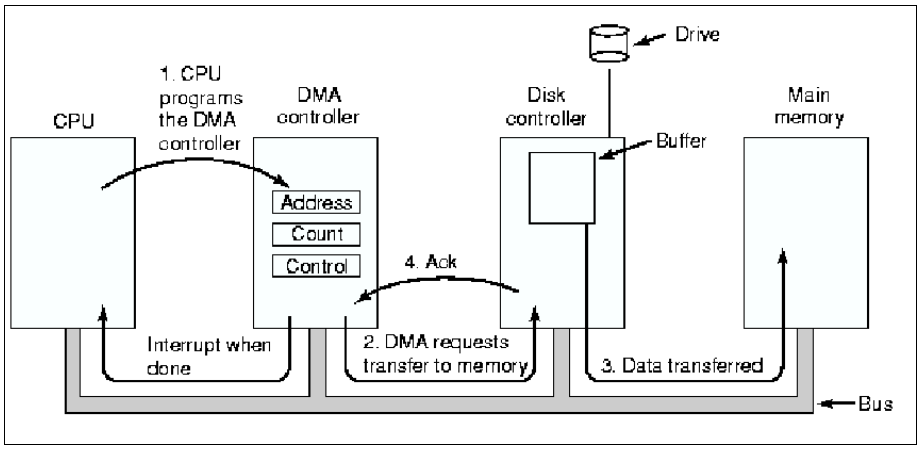


Figure V-14: Funcionamiento de una DMA

Los pasos son:

1. La CPU programa la controladora de DMA.
2. La CPU indica a la controladora de E/S que empieza a leer del dispositivo.
3. Cuando hay datos correctos en el buffer de la controladora la DMA pone en el bus la dirección donde la controladora escribirá.
4. La controladora detecta la petición de escritura y la realiza.
5. Cuando la controla acaba usa el bus para enviar un acuse de recibo.
6. La DMA lo detecta y modifica los registros de dirección de memoria y conteo.
7. Si el valor del conteo es 0 se envía a la CPU una interrupción indicando la finalización. En otro caso vuelve a 3.

La transferencia entre el dispositivo y memoria se pueden hacer **palabra a palabra** (usando el **robo de ciclo**) o **por ráfagas**. El robo de ciclo consiste en que la controladora de E/S “roba” el bus de sistema a la CPU para hacer la transferencia:

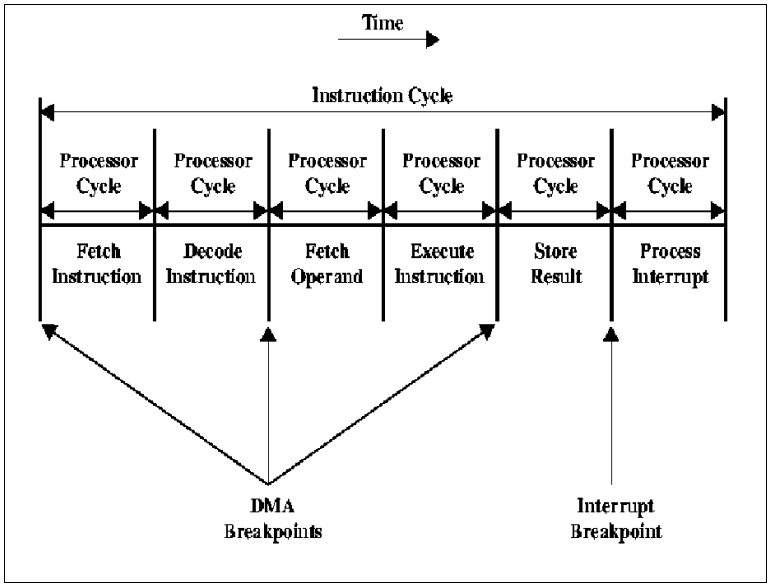


Figure V-15: Robo de clclos

La técnica por ráfagas se basa en que cuando nos apoderamos del bus podemos aprovechar para enviar más de una palabra.

Si es la controladora E/S la que escribe en memoria hablamos de una DMA en **modo de sobrevuelo**. Otra opción es que lo haga la propia DMA (más ciclos de reloj).

Observe que esta técnica hace **uso de un buffer** por parte de la controladora de E/S, esto es necesario ya que:

- Cotejar la validez de los datos antes de hacer la transferencia.
- El disco transfiere de forma continua datos a la controladora independientemente de si la controladora está lista o no. ¿Qué pasa si la controladora no se puede hacer con el bus de sistema para enviar los datos?

Para finalizar, y si retomemos el ejemplo de la impresora, la CPU ejecutaría lo siguiente:

```
// Al efectuar la llamada al sistema
copiar_de_usuario(buffer_usr, buffer_kernel, cuenta);
preparar_controladora_DMA();
planificar proceso();


// Procedimiento de servicio de la interrupción
acusar_interrp();
desbloquear_usr();
volver_de_interrup();
```

Observe que el procedimiento de servicio de la interrupción sólo se ejecutará cuando acabe la operación de entrada salida.

Los discos duros actuales usan diferentes modos DMA para hacer las transferencias entre memoria y disco. Los modos actuales, denominados Ultra DMA, se caracterizan por enviar el doble de datos por ciclo de reloj con lo que conseguimos el doble de ancho de banda.

3. Objetivos del software de entrada/salida

El software de entrada salida persigue los siguientes objetivos:

- **Eficiencia**: las operaciones de E/S suelen ser el cuello de botella dentro de un sistema, es deseable que sean lo más eficientes posibles (uso de técnicas de *buffering* y *spooling*).

- **Seguridad y protección**: ha de procurar la protección de los dispositivos (acceso a los dispositivos por parte de los usuarios). También debe conseguir seguridad del sistema (los errores producidos por uno de los usuarios no ha de afectar al funcionamiento del resto del sistema).

- **Independencia de dispositivos**: ha de ser posible escribir programas que puedan utilizarse para cualquier dispositivo de E/S sin tener que modificar los programas para cada tipo de dispositivos. Se intenta **desvincular totalmente los programas de sus entrada/salidas**. Un ejemplo claro lo tenemos en UNIX, donde podemos hacer que un mismo programa pueda ejecutar con distintos dispositivos de E/S cada vez que lo ejecutamos (mediante la redirección). Llevado al máximo extremo permite que un programa se pueda transportar entre máquinas que utilizan el mismo SO (e incluso diferentes). A esto se denomina **portabilidad**.

  Esta independencia de dispositivo la podemos ver a diferentes niveles:

  - **Independencia del juego de caracteres**. La representación interna de los datos debe ser transparente al programador.

  - **Independencia del periférico usado**. La podemos ver a tres niveles:

    - Independencia entre **diferentes tipos de la misma clase** de dispositivo (cualquier tipo de impresora).

    - Independencia entre dispositivos de **diferentes clases** (imprimir por impresora o pantalla).

    - Independencia de la **unidad de transferencia**. El número de caracteres con los que el programa hace la E/S ha de ser independiente de si este es un terminal que lee caracteres de uno en uno, o de si es un disco que lee un bloque de 1K (flujo o bloques).

- **Manejo de errores**: deben manejarse lo más cerca posible del hardware.

- **Plug & Play**: permitir conectar dispositivos solventando automáticamente su instalación.

- Diferencia entre **transferencias síncrona y asíncronas**: casi todas las operaciones de E/S suelen ser asíncronas. Los programas de usuario son más fáciles de escribir si suponemos bloqueo en las operaciones de E/S. El SO debe conseguir que operaciones no bloqueantes lo parezcan.

- **Dispositivos compartidos y dedicados**: hay diferentes problemas si un mismo dispositivo de E/S es usado por más de un usuario a la vez o solamente por uno.

4. Principios del diseño del software de entrada/salida

Para lograr los objetivos descritos, en especial el de la independencia de dispositivos,los diseñadores se basan en los siguientes principios:

- **Unificación de Operaciones**: proporcionar un conjunto reducido de operaciones (llamadas al sistema) suficientemente generales como para servir para todos los tipos de dispositivo (open, close, read, write,...). Hay ocasiones en las que se puede conseguir, pero para ciertos dispositivos es necesario tener operaciones específicas (ioctl, fcntl)
- **Dispositivos Virtuales**: los programas deben trabajar con dispositivos virtuales en vez de hacerlo con dispositivos físicos. El SO asignará a cada dispositivo virtual un dispositivo físico. Será el SO el que conocerá a través del dispositivo virtual con qué dispositivo físico está trabajando el programa. En el caso de LINUX se tiene un fichero asociado a cada dispositivo; también disponemos de la tabla de canales para un proceso, donde se asignan los dispositivos. El SO ha de proporcionar una manera de poder asociar un dispositivo virtual con uno físico y de poder deshacer dicha asociación (operaciones de asignar, desasignar).
- **Redireccionamiento**: un SO permite redireccionar los dispositivos virtuales si estos pueden ser asignados antes de que un proceso entre en ejecución, de tal forma que éste ya se los encuentre abiertos. Este mecanismo permite cambiar de una ejecución a otra los dispositivos físicos con los que trabaja por defecto un programa, sin que haya que modificar su código.

5. Capas del software de entrada/salida

El software de entrada/salida es lo suficientemente complejo como para no tratarlo de manera única. Al igual que pasaba en el diseño del núcleo del sistema operativo, se busca dividir este software en diferentes capas o niveles de manera que las capas superiores dependan de las inferiores.

Teniendo el flujo de una operación de entrada/salida desde que es generado por una aplicación hasta que es completada, podemos hablar de tres capas:

- Software de entrada/salida en el espacio de direcciones del usuario.
- Software de entrada/salida independiente del dispositivo
- Software de entrada/salida dependiente del dispositivo

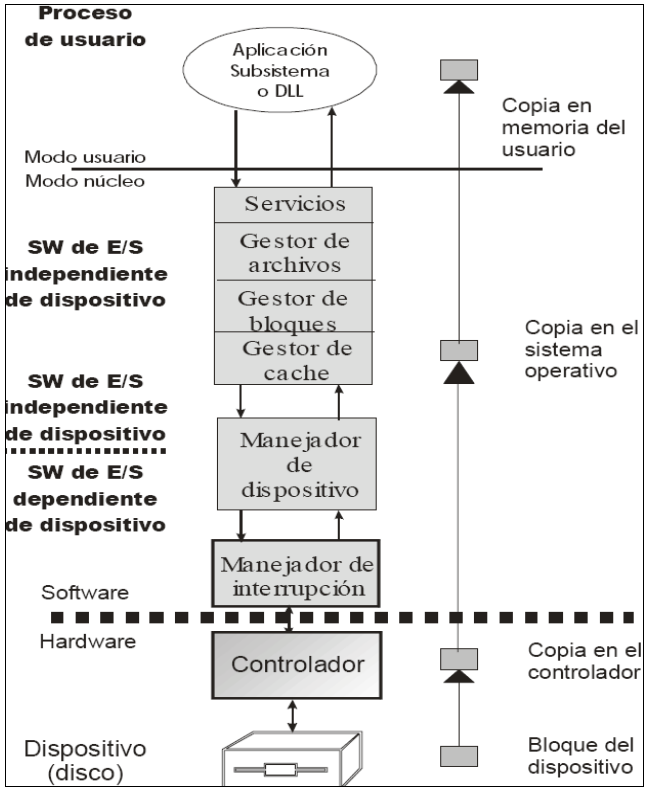


Figure V-16: Capas del software de entrada/salida

El gestor de interrupciones se puede considerar software de entrada/salida dependiente del dispositivo pero debido a su importancia se tratará como capa independiente.

5.1. Gestión de interrupciones

Como ya hemos vistos los SO deben de estar preparados para recibir interrupciones provenientes de los dispositivos de E/S.

Una vez que la interrupción es aceptada se suceden los siguientes pasos:

1. Guardar los **registros** que no haya guardado aún el hardware.
2. Preparar un **contexto** para el procedimiento del servicio de interrupción.
3. Preparar una **pila** para el procedimiento de servicio de interrupción.
4. Enviar un **acuse de recibo** a la controladora de interrupción.
5. Copiar los registros desde donde se guardaron a la tabla de procesos.
6. **Ejecutar** el procedimiento de **servicio de interrupción** que interactuará con la controladora.
7. **Escoger el proceso** que se ejecutará a continuación.
8. **Preparar el contexto** para que el proceso se ejecute.
9. **Cargar** los **registros** del proceso.

10. Comenzar la **ejecución** el proceso.

5.2. Gestores de dispositivos, los drivers

Las controladoras de E/S tienen diversos registros que sirven para comunicarse con el dispositivo. El número de registros y la naturaleza de los comandos varían de forma radical de un dispositivo a otro. Debido a esto cada dispositivo de E/S necesita **código específico** que sirva para controlar el dispositivo. A este código se le denomina controlar de dispositivo (*device driver*). El kernel debe tener embebido un driver para cada uno de los periféricos presentes en el sistema.

Funciones

En la siguiente gráfica se muestra de forma resumida el funcionamiento de cualquier driver de un dispositivo:

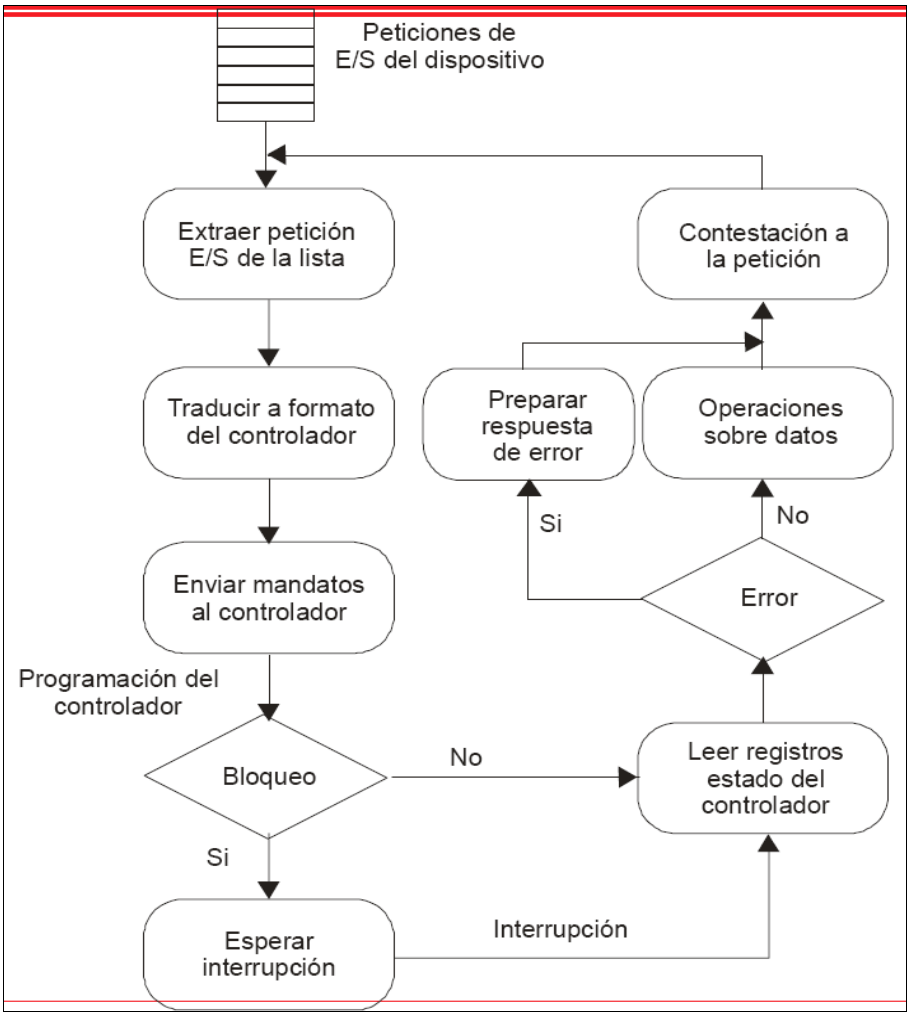


Figure V-17: Funcionamiento general de un driver

Teniendo en cuenta lo indica en esta figura podemos decir que las principales funciones de un driver son:

- Definir **características** del **dispositivo periférico** para el resto del SO y de los procesos de usuario.

- **Asignar** los **valores iniciales** adecuados a los registros asociados al dispositivo en el momento de arranque o encendido del sistema.
- Permitir que un proceso pueda **acceder** a un **dispositivo**, o bien **retirarle** dicho **permiso**.
- **Procesar** las operaciones de **E/S** solicitadas por cualquier proceso que tenga permitido el acceso al dispositivo.
- **Cancelar** todas las operaciones de **E/S** en el momento que se considere necesario.
- **Procesar** todas las **interrupciones** hardware generadas por el dispositivo.
- Tratar los **errores** y **estado del dispositivo** y comunicárselo a los niveles superiores del sistema operativo para que éste se lo comunique a su vez al usuario.

Características

De forma resumida podemos decir que algunas de las características más sobresalientes de un driver son:

- Un driver no es un proceso sino un conjunto de **tablas** en las que se aloja información y una serie de **rutinas**. Llamando a estas rutinas tendremos acceso a las diferentes funciones del controlador.
- Los drivers deben de ser **reentrantes**, deben considerar la posibilidad de ser interrumpidos antes de acabar un servicio.
- Los drivers sólo puede ejecutar un número limitado de llamadas al sistema.
- Es propio de cada fabricante y sistema operativo.
- Se ejecuta en modo kernel ya que necesita acceder a los registros de la controladora. es necesario que el controlador forme parte del kernel del SO. Si el driver tiene errores la estabilidad del sistema se puede ver muy afectada.
- Presenta un interfaz bien definido entre él y el resto del núcleo del que es integrante. Suelen existir dos interfaz diferentes una para los drivers de bloques y otra para los de caracteres-.

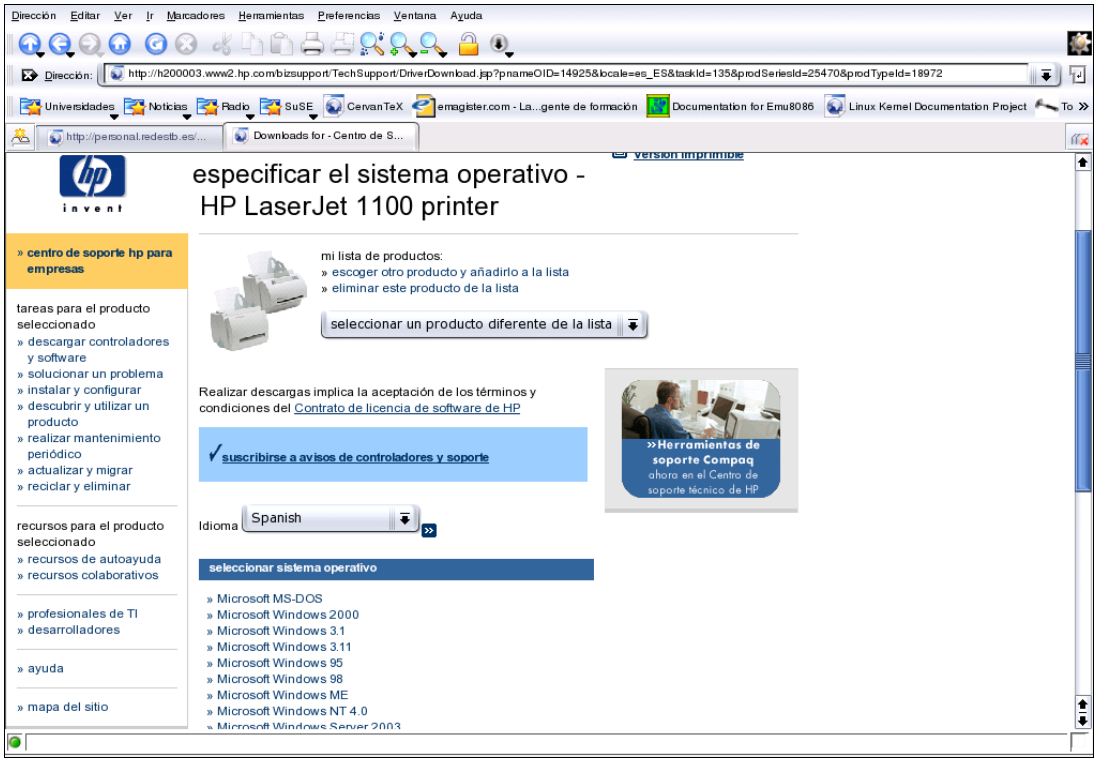


Figure V-18: Los drivers son proporcionadas por los fabricantes

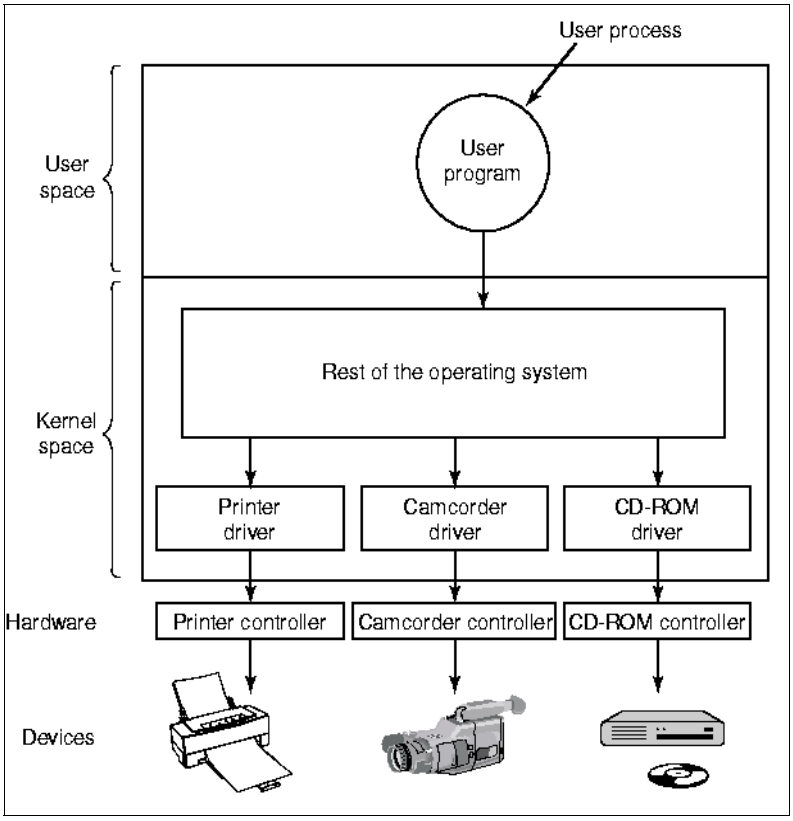


Figure V-19: Los drivers son proporcionadas por los fabricantes

- En un principio la carga de los controladores era estática pero actualmente se hace bajo demanda, sólo se carga cuando se necesita.
- Un driver puede atender varias unidades de un mismo dispositivo. Las políticas de atención pueden ser en serie o en paralelo.

5.3. Software independiente del dispositivo (Servicios de E/S)

Es aquel software cuyas funciones esenciales son:

- Interfaz uniforme para controladores de dispositivos (drivers).
- Manejo de *buffers*.
- Informe de errores.
- Asignar y liberar dispositivos dedicados.
- Proveer un tamaño de bloque independiente del dispositivo.

Dentro de este software podemos incluir al sistema de archivos (que lo veremos en el próximo tema), el gestor de bloques, la cache de bloques y el interfaz superior de los manejadores de dispositivo.

Interfaz uniforme para controladores de dispositivo

No hay que perder de vista que uno de los objetivos del software de entrada/salida era conseguir que todos los dispositivos de entrada/salida y sus controladores tuvieran un aspecto similar de cara al sistema operativo.

Para ello se busca un interfaz común entre el sistema operativo y los dispositivos intentando reducir los tipos de dispositivos e incluso tratándolos a estos de la misma manera.

Para lograr esto las operaciones de entrada/salida encapsulan el comportamiento de los dispositivos de forma genérica. Forma en que se nombran los dispositivos de E/S: se encarga de establecer la correspondencia entre los enlace simbólicos y el dispositivo(en Unix inodos):

Será la capa dependiente del hardware la que esconda las diferencias entre los controladores de entrada/salida del núcleo.

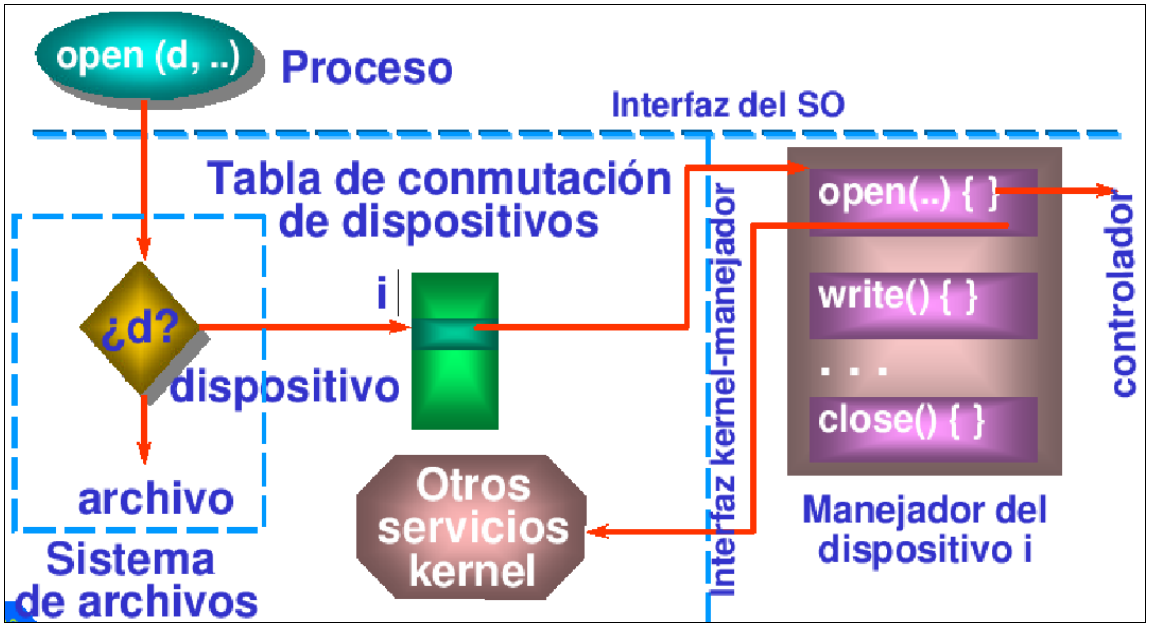


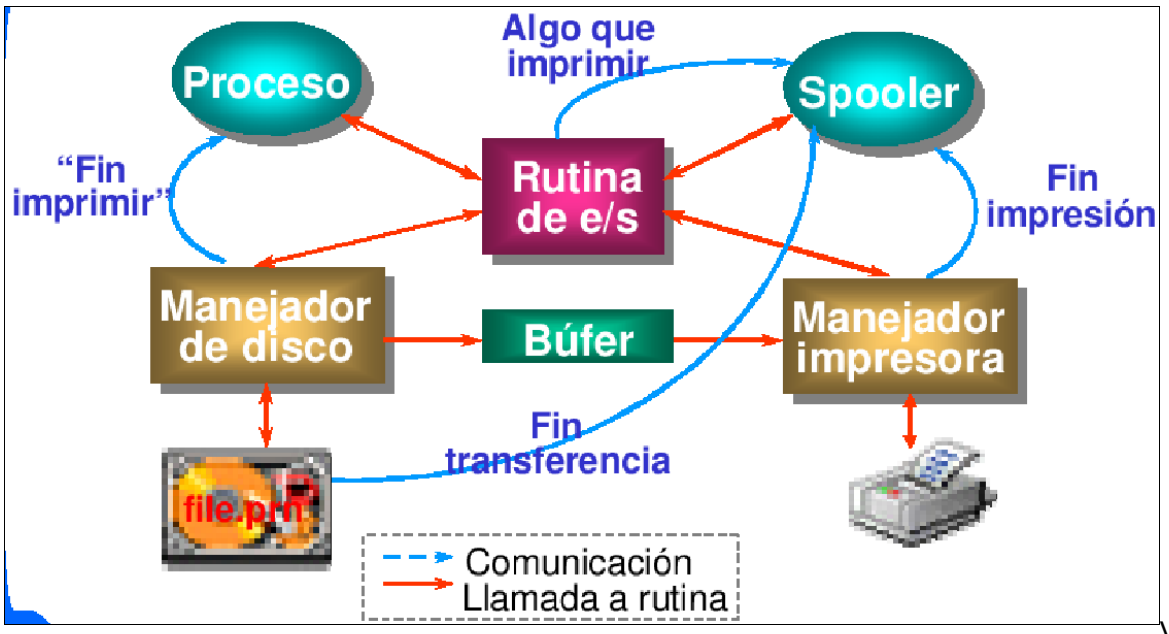
Figure V-20: Los drivers son proporcionadas por los fabricantes

Forma en que se nombran los dispositivos de E/S: se encarga de establecer la correspondencia entre los enlace simbólicos y el dispositivo(en Unix inodos):

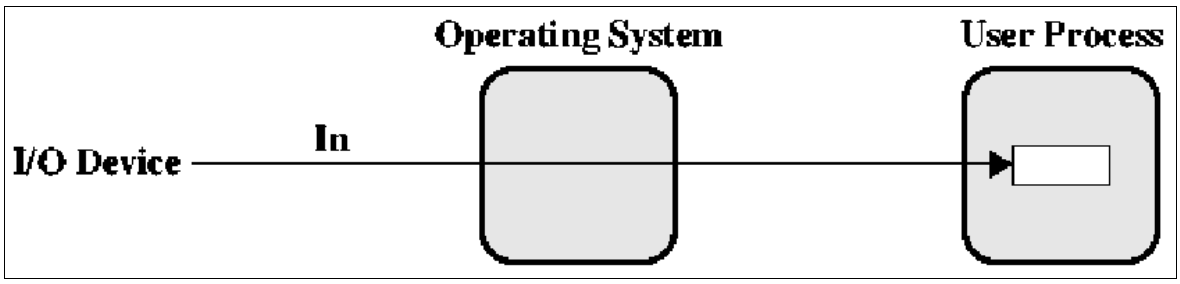
- Protección de dispositivos, en Unix y Windows 2000 las reglas normales de protección de archivos también son válidas.

Manejo de buffers

El subsistema de entrada salida hace uso de los buffers. El esquema general de funcionamiento es el siguiente:



Veamos por qué es necesario el uso de *buffers*. Cuando un proceso de usuario realiza una operación de lectura los datos se copian dentro del espacio de direcciones del proceso del usuario:



Esta solución presenta los siguientes desventajas:

- El programa se queda colgado esperando que acabe la operación de E/S.
- La E/S interfiere en las decisiones de intercambio (no se puede realizar hasta que acabe la operación).

Para solucionar estos problemas podemos llevar a cabo las transferencias de entrada por adelantado a las peticiones y realizar las transferencias de salida un tiempo después de hacer la petición. Esta técnica se conoce como buffering o almacenamiento intermedio. Lo que se pretende es mantener tanto al procesador como a los dispositivos de E/S ocupados. Hay distintas aproximaciones:

- Memoria intermedia sencilla: el SO reserva parte de su espacio a la operación. Cuando el buffer está lleno se pasan los datos al espacio del usuario.

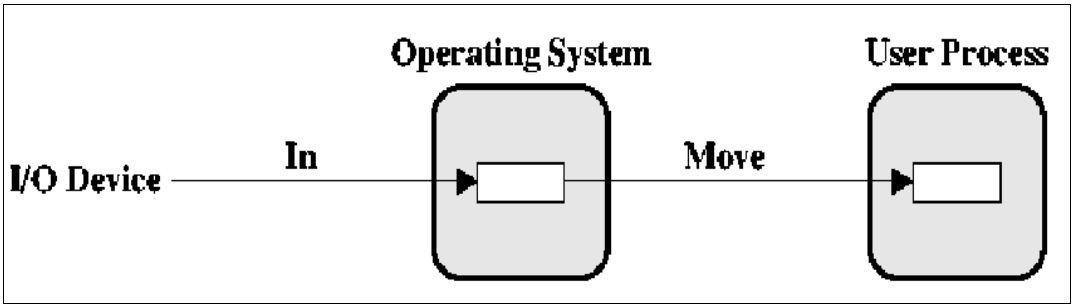


Figure V-21: Buffer simple

- Memoria intermedia doble: un proceso puede transferir datos hacia memoria intermedia mientras que el sistema operativo vacía el otro.

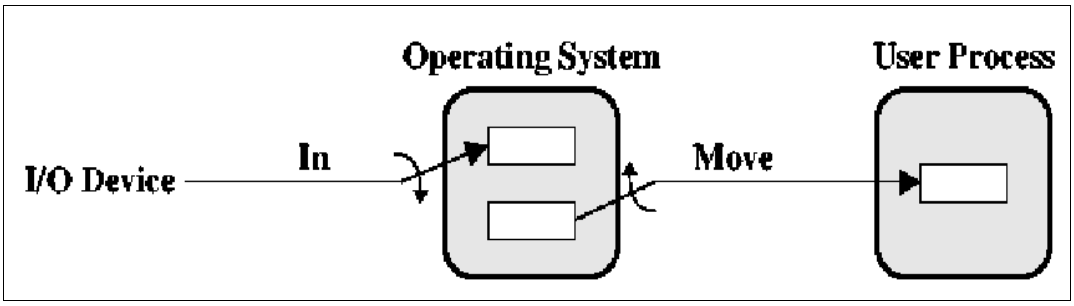


Figure V-22: Buffer doble

- Memoria intermedia circular: se usa más de dos memorias intermedias.

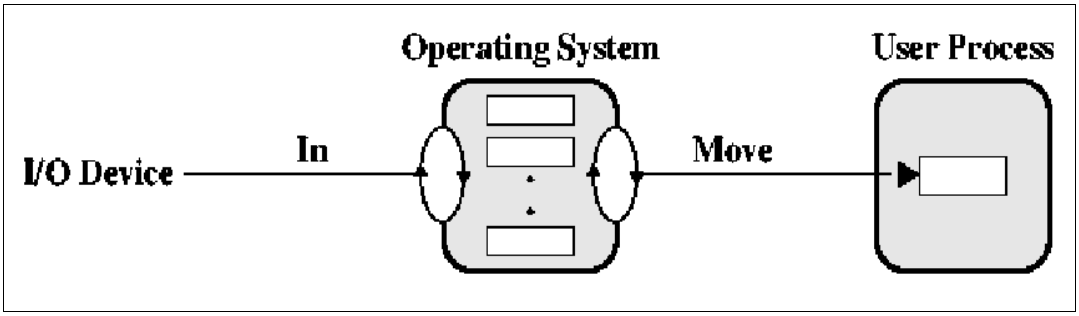


Figure V-23: Buffer circular

El almacenamiento intermedio soluciona los problemas de picos en la demanda de E/S. No existe un tamaño de memorias secundarias intermedias que permita a un dispositivo mantener un ritmo indefinido para un proceso cuando la demanda media del proceso es mayor que la que el dispositivo admite. Si se usan muchos buferes sucesivos puede existir una merma en el rendimiento.

Informe de errores

Los errores son muy comunes en la E/S y su manejo corresponde al controlador apropiado pero el marco general del manejo de errores es independiente del dispositivo.

Clases de error:

- de programación (intentar escribir en un dispositivo de entrada, paso incorrecto de parámetros): en esos casos de devolverá un código de error
- errores reales de E/S (escribir en un bloque de disco dañado): el controlador determinará qué hacer.
- Lo que hace el software cuando reciba el error dependerá de la naturaleza del error.

Asignación y liberación de dispositivos dedicados

Algunos dispositivos sólo pueden ser utilizados por un proceso a la vez. El SO examinará las solicitudes de uso de dispositivos y aceptarlas o rechazarlas, dependiendo de si el dispositivo solicitado está o no disponible. ¿Cómo controlar esto?

- Obligar al proceso que haga un **open** y un **close** sobre un fichero que representa al dispositivo.
- Contar con mecanismos para solicita y liberar recursos dedicados (el proceso se bloquea a la espera de concederse el recurso).

Proveer un tamaño de bloque independiente del dispositivo

Diferentes discos duros pueden tener tamaños de sector distintos. El software independiente del dispositivo debe ocultar estas diferencias presentando un tamaño de bloque único.

5.4. Software de E/S en el espacio de usuario

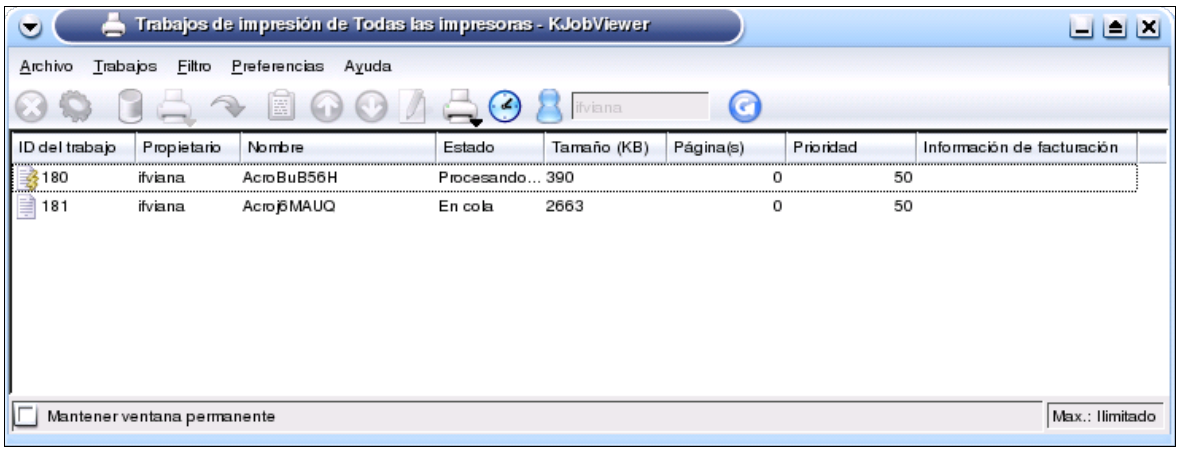
Una pequeña parte del software de E/S se ejecuta fuera de kernel:

- **Librerías** que estarán formadas por procedimientos que colocarán los parámetros en un lugar apropiado para realizar la llamada al sistema y quizás realicen alguna conversión de formato.
- **Programas** como el sistema de spooling (Spooling simultaneous Peripheral Operation On-line) que es una manera de manejar dispositivos dedicados en sistemas con multiprogramación.

El spooling se suele usar con las impresoras. El spooling funciona de la siguiente manera:

- Se crea un proceso especial denominado **demonio** y un directorio especial denominado **directorio de spool**.
- Cuando un proceso desea imprimir el fichero a imprimir se guarda dentro del directorio de spool.
- El demonio comprobará el spool e irá enviando los distintos ficheros al dispositivo siguiendo algún criterio de planificación.

Existe multitud de software de spool para impresoras como: CUPS, LPD, LPRng, PPR, etc.



6. Almacenamiento secundario

Existen diversos dispositivos de entrada/salida, un grupo de ellos los formas los denominados dispositivos de almacenamiento secundario que se caracterizan por ofrecen:

- Lectura y escritura aleatorias.
- Almacenamiento no volátil.
- Alta velocidad de acceso, debajo de la RAM en la jerarquía.
- Susceptibles de ser soporte para el sistema de archivos.

Los elementos principales de este sistema:

- Discos: es interesante conocer su estructura y cómo se gestionan.
- Manejadores de disco que controlan todas las operaciones sobre los discos. Es especialmente importante la planificación de peticiones.

7. Discos

Los discos constituyen el soporte para el sistema de intercambio del gestor de memoria virtual o para el sistema de archivos.

Se pueden clasificar de diversas maneras atendiendo a diversos criterios. Según la interfaz de su controlador:

- Dispositivos SCSI (*Small Computer System Interface*).
- Dispositivos IDE (*Integrated Drive Electronics*).
- Dispositivo SATA (*Serial ATA*).

Y si tomamos como referencia la tes tecnología que emplean:

- Discos duros o magnéticos (*Winchester*).
- Discos ópticos.
- Discos extraíbles.

Discos Magnéticos

Los discos magnéticos se estructuran físicamente de la siguiente manera:

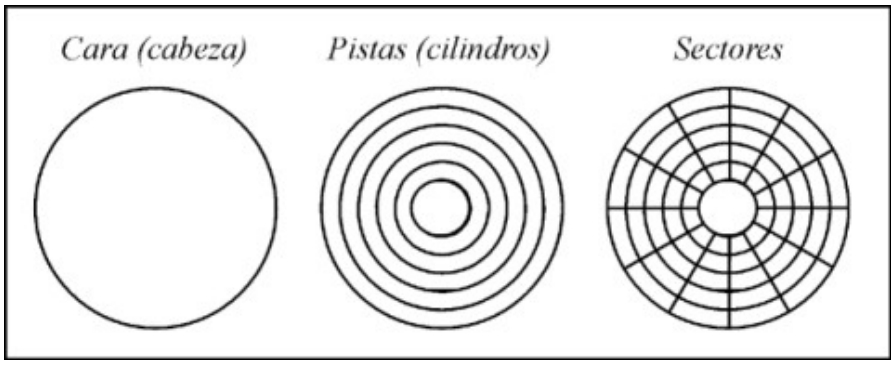


Figure V-24: Estructura interna de un disco magnético

- **Cara** (cabeza): cada una de los discos que son accesibles, empiezan por la 0.
- **Pista** (track): Divisiones concéntricas del disco que son accesibles por la cabeza de lectura / escritura, empiezan por la 0. Actualmente la densidad de almacenamiento es constante por lo que las pistas exteriores
- **Sector**: divisiones lógicas radiales de un disco, para poder localizar mejor la información, empiezan por el 0. Suelen tener un tamaño de 512 bytes. Para crear estos sector se debe realizar un formateo físico o de bajo nivel. El resultado final es que cada pista se divide en sectores, se identifica a cada uno de ellos y se añade espacio para la detección y corrección:

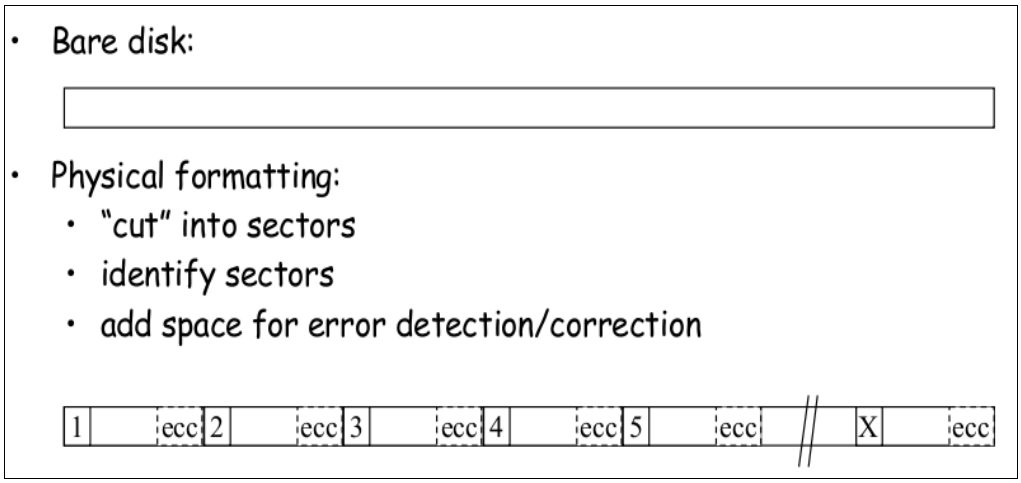


Figure V-25: Formateo físico

- **Cilindro**: todas las pistas concéntricas de un disco.

Para calcular la capacidad completa del disco basta con multiplicar los valores máximos de cilindros, caras, sectores por pista y tamaño del sector.

Al la tupla formada por (cilindros, cabezas y sectores por pista) se le denomina **geometría del disco** o **CHS**.

En las computadoras basadas en Pentium el valor máximo de estros tres parámetros suele ser (65535, 16, 63). Los cilindros y sectores se numeran a partir del 1 y las caras en 0.

Ejemplo 1: con estos valores y suponiendo sectores de 512 bytes por sector, ¿cuál puede ser el tamaño máximo de un disco? 31.5 GB.

Los discos duros de mayor tamaño (> 31.5 GB.) usan **direccionamiento por bloque lógico**, los sectores se numeran consecutivamente a partir del cero sin tener en cuenta la geometría.

Ejemplo 2: el disco duro ST33221A de Seagate tiene las siguientes especificaciones: cilindros = 6.253, cabezas = 16 y sectores = 63. El número total de sectores direccionables es, por tanto, 6.253*16*63 = 6.303.024 sectores. Si cada sector almacena 512 bytes de información, la capacidad máxima de este disco duro será de 6.303.024 sectores * 512 bytes/sector = 3.227.148.228 bytes ~ 3 GB

Ejemplo 3: Parámetros de un disco flexible y un disco duro:

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μsec

Actualmente la densidad lineal de almacenamiento es constante por lo que las pistas exteriores tienen más sectores que las interiores. Además en las controladores actuales se pueden almacenar pista completas y, si el controlador es inteligente y se encarga de varios dispositivos, puede efectuar operaciones de forma solapada.

Funcionamiento de un disco duro

Una representación esquemática de un disco duro se puede ver como varios platos de metal sujetos por un eje central:

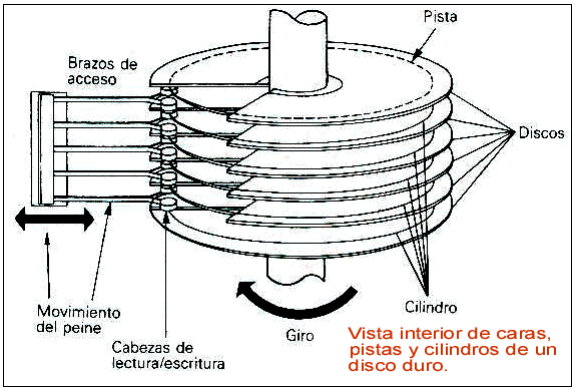


Figure V-26: Sectores, bloques y páginas

Entre cada plato (leyendo cada una de las dos caras) existe un brazo que emite pulsos magnéticos:

Figure V-27: Sectores, bloques y páginas

Los platos giran a una velocidad que puede oscilar entre las 5600-10000 rpm, en sentido contrario a las agujas del reloj. En los extremos de estos brazo hay unas cabezas lectoras (escritoras) que leen (escriben) la porción de disco que hay debajo. Las cabezas lectoras se mueven del centro a los extremos o viceversa.

Figure V-28: Grabación en espiral

En los discos duros modernos el número de sectores por pista varía. La superficie del mismo se divide en zonas, las zonas más externas tienen más sectores:

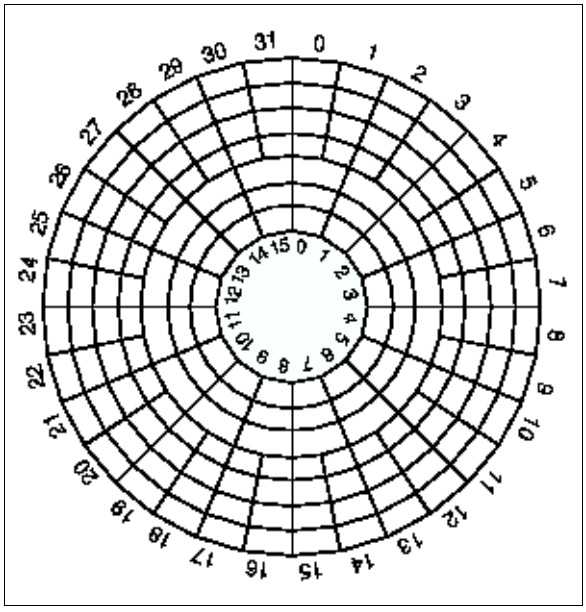


Figure V-29: Grabación en espiral

Para evitar que el SO sea consciente de estas peculiaridades, los discos modernos presentan una geometría virtual:

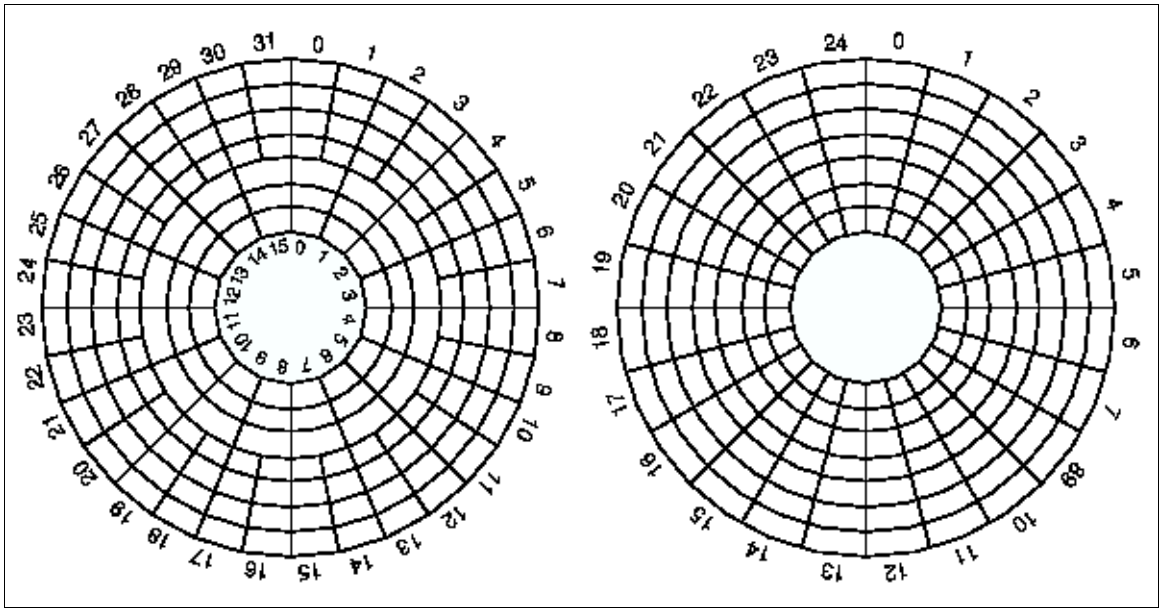


Figure V-30: Grabación en espiral

La conversión entre la geometría virtual y la física la realiza la controladora (LBA).

Medidas

El tiempo que se tarda en leer la información desde el disco (también denominado como **tiempo de acceso**)se divide en:

$$t_{acceso}=t_{busqueda}+t_{latencia}+t_{transferencia}$$

donde:

- **Tiempo de latencia (T_r)**: tiempo que toma a los datos girar desde la posición en la que se encuentran hasta la posición en la que está la cabeza de lectura/escritura. El brazo móvil se mueve hacia delante y hacia atrás. Mientras se encuentra en una posición se puede leer toda la pista (cilindro) de un disco. Se define como:

  $$T_{latencia}=1/2\mathrm{r}$$

  Donde **r** viene dado en revoluciones por unidad de tiempo.

- **Tiempo de búsqueda (T_s)**: tiempo que tarda el brazo móvil en desplazarse hasta la nueva pista (acceso a la pista):

  $$T_{busqueda}=m\cdot n+s$$

  Donde **m** es el número de pistas atravesadas, **n** es el tiempo que se tarda en pasar cada pista y **s** es el tiempo de configuración o arranque.

- **Tiempo de transferencia**: tiempo que se tarda en transferir la cantidad de datos deseada desde el disco a la memoria:

  $$T_{transferencia}=b/rN$$

Donde **b** es el número de bytes a transferir y **N** el número de bytes por pista.

A parte de estos tiempo también hay que tener en cuenta el tiempo que hay que esperar por el dispositivo y por el canal:

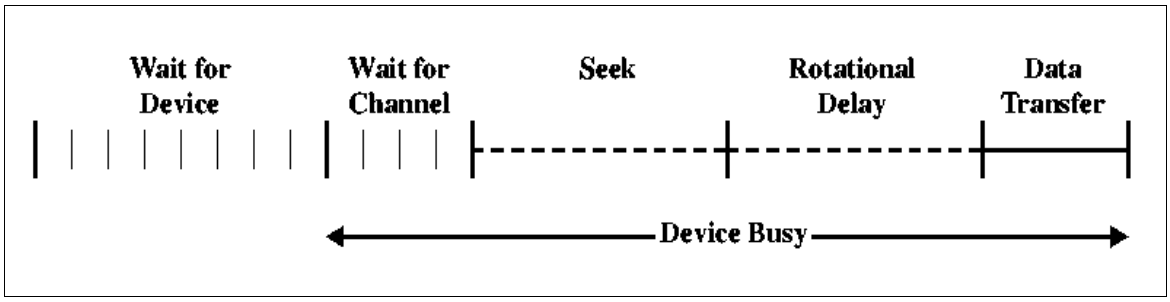


Figure V-31: Grabación en espiral

Normalmente un archivo se almacena diseminado en pistas, sectores y cilindros o sea se graba en las caras de los distintos platos simultáneamente, porque la estructura que sostiene los brazos con sus cabezas de lecto-escritura mueve todo el conjunto de cabezas al mismo tiempo.

Discos ópticos

Son un tipo de soporte de almacenamiento que se caracterizan por tener una densidad de grabación mayor a la de los discos magnéticos-Dentro de los discos ópticos veremos:

- CD-ROM
- CD grabables
- CD regrabables
- DVD

CD-ROMs

En 1980, Philips y Sony desarrollaron el CD para grabar música. Los datos técnicos se publicaron en el Libro Rojo (norma ISO 10149).

Entre otras cosas esta norma especifica un diámetro de 120 mm, un grosor de 1.2 mm y un agujero central de 15 mm.

Para crear CD-ROM se usa un láser de alta potencia para quemar agujeros de 0.8 micras de diámetro (también llamados fosos o *pits*) en un disco maestro de vidrio recubierto. A partir de este disco maestro, se prepara un molde con protuberancias donde estaban los agujeros del láser. En este modelo se inyecta resina de policarbonato fundida, luego una capa de aluminio y finalmente una resina protectora.

Para leer un CD se usa un diodo láser de baja potencia que dirige un haz de luz a la zona de fosos y llanos. Según la cantidad de luz que se refleje el diodo distinguirá entre un foso y un llano. Una transición foso/llano o llano/foso codifica un 1 y su ausencia un 0. Los fosos y los llanos se graban en una sola espiral continua de 5.6 Km de longitud:

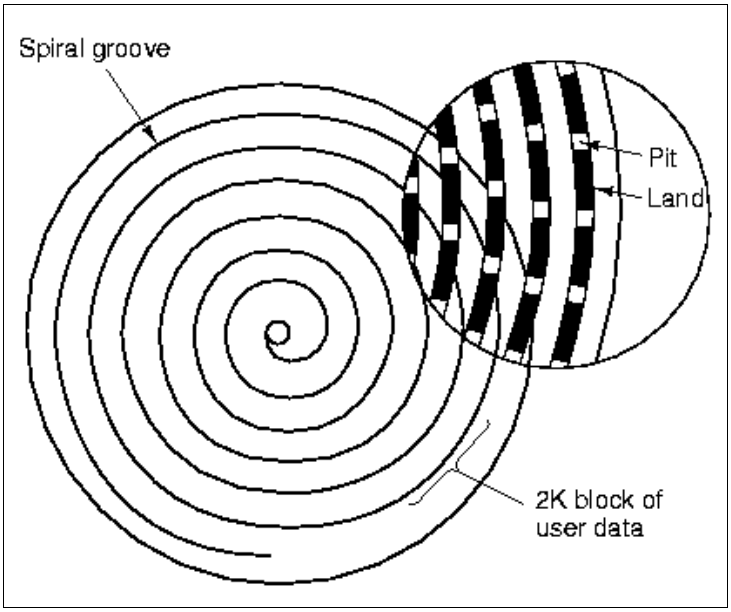


Figure V-32: Grabación en espiral

Para que la música se reproduzca a ritmo uniforme es necesario mantener la velocidad linea constante lo que implica que la velocidad de rotación se reduce conforme nos alejamos del centro del CD. La música se almacena siguiendo el siguiente formato de tramas:

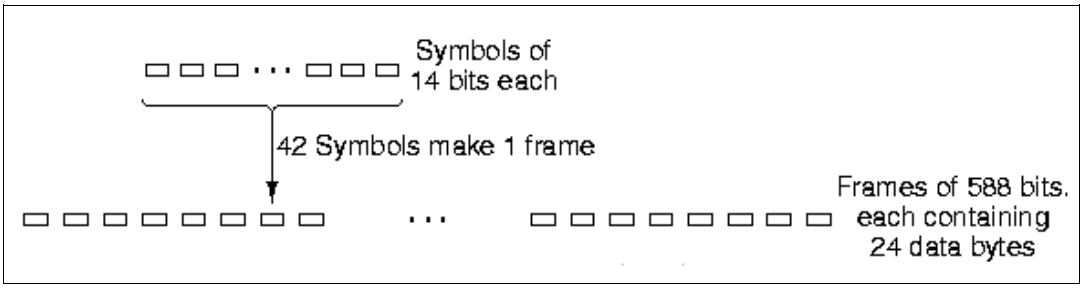


Figure V-33: Estructura de las tramas en un CD

En 1984 Philips y Sony publican el **Libro Amarillo** que define la norma para almacenar datos ⇒aparecen los CD-ROMS (*Compact Disk Read Only Memory*). Por compatibilidad con los CD se mantienen las mismas características técnicas. Básicamente se define el formato de los datos y la capacidad de corrección de datos. Dicho formato es:

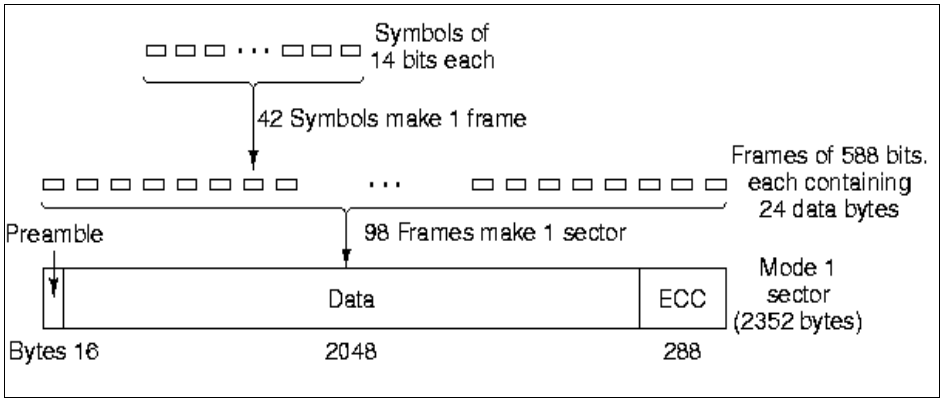


Figure V-34: Formato de CD-ROM

Se añade el concepto de sector (agrupaciones de 98 tramas) y el de preámbulo que indica el comienzo del sector, el número de sector y el modo. Hay dos modos el modo 1 para usar corrección de errores y el 2 para no usarla y combinar los campos de datos y de error en uno único de datos. Se usan esquemas de corrección dentro de un símbolo, dentro de una trama y dentro de un sector.

En 1986 Philips presentó el **Libro Verde** que añadía la capacidad de poder intercalar gráficos, audio, vídeo y datos en el mismo sector. Por último, para poder usar un mismo CD en diferentes computadoras era preciso usar un mismo sistema de ficheros, nace el ISO 9660. Define 3 niveles:

- **Nivel 1**: nombre de archivos de 8 caracteres (con 3 más opcionales para la extensión), los nombres contienen sólo letras mayúsculas, dígitos y el carácter. Anidamiento de directorios de hasta 8 niveles.
- **Nivel 2**: Nombres de hasta 32 caracteres La extensión Rock Ridge permite nombres muy largos, UID, GID ...
- **Nivel 3**: además permite ficheros no contiguos.

CDs grabables

En los 90 aparecen unidades CD-R (CD-Recordable) que permitían grabar datos usando pequeñas unidades de grabación. Los CD-R son como los CD salvo:

- que tienen un surco de 0.6 micras para guiar el láser durante la escritura.
- Suelen tener un color dorado y no plateado (no usan aluminio en la capa reflectante).
- La reflectividad entre fosos y llanos se simula mediante el uso de una capa de colorante:

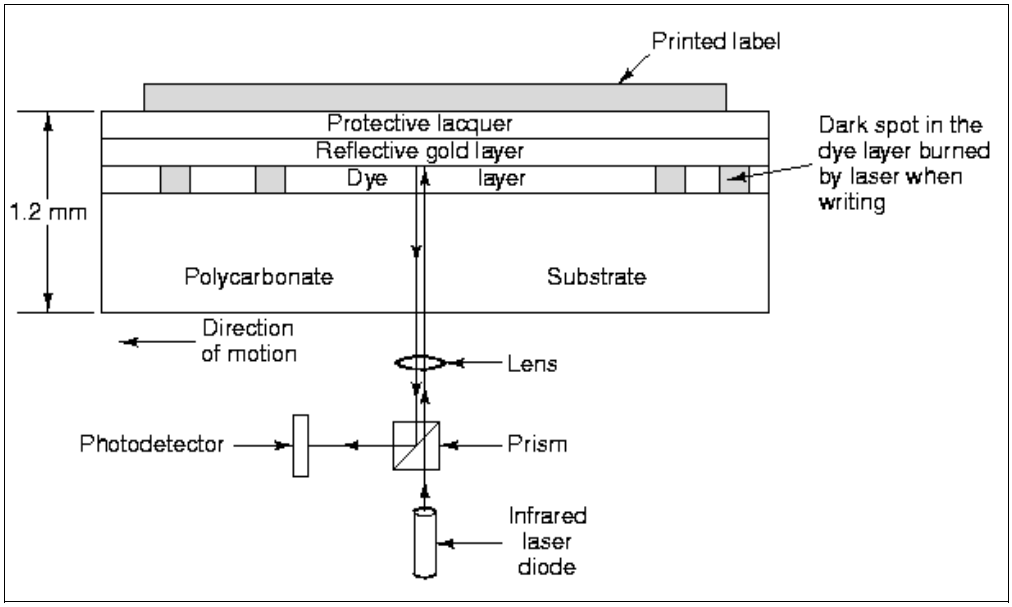


Figure V-35: Formato de CD-ROM

La especificación del CD-R está recogida en el **Libro Naranja** publicado en 1989. En esta especificación también aparece recogido el formato CD-ROM XA, que permite escribir los CD-R de forma incremental. Un grupo de sectores consecutivos escritos en una misma ocasión es una pista de CD-ROM. Antes cada CD-ROM tenía una única VTOC (*Volume Table Of Contents*), ahora cada pista tiene su propia VTOC que puede incluir información de pistas anteriores (si no la incluye se da la sensación que los datos se han perdido). Las pistas pueden agruparse en sesiones dando lugar a CD-ROM multisesión.

CDs regrabables

Los CD-RW (CD-ReWritable) permiten borrar su contenido. Son como los CD-R pero usan un colorante basado en una aleación de plata, indio, etc. que puede estar en dos estados: cristalina y amorfa Las unidades CD-RW tienen un láser con tres niveles de potencia:

- El primer nivel funde la aleación pasando de estado cristalino a amorfo (simula un foso).
- El segundo nivel funde la aleación para pasarla a un estado cristalino (simula llano).
- El tercer nivel sirve para la lectura.

DVD

La necesidad de tener discos ópticos con mayor capacidad hizo que 10 compañías de electrónica, en estrecha relación con los principales estudios de Hollywood, desarrollaran el DVD (Disco Digital Versátil).

Aunque inicialmente hubo dos formatos (MMCD liderado por Sony y SD con el respaldo de Toshiba) actualmente hay un único estándar.

Las unidades lectoras de DVD necesitan un segundo láser para poder CD tradicionales.

Hay diferentes formatos de DVD:

- DVD5: un solo lado, una sola capa de Philips y Sony.

- DVD9: un solo lado, doble capa de Philips y Sony.
- DVD10: dos lados, una sola capa de Toshiba y Time Warner.
- DVD18: Dos lados, doble capa de Toshiba y Time Warner.

DVD-R (hay que formatear DVD y cerrarlo pero es más compatible) y DVD+R son dos estándares de grabación.

La tecnología de doble capa tiene una capa reflectante abajo, y más arriba una capa semireflectante. Dependiendo del lugar donde se enfoque el láser, el haz rebotará en una capa o en otra.

Los discos de dos lados se fabrican tomando dos discos de un solo lado y pegándolos reverso con reverso.

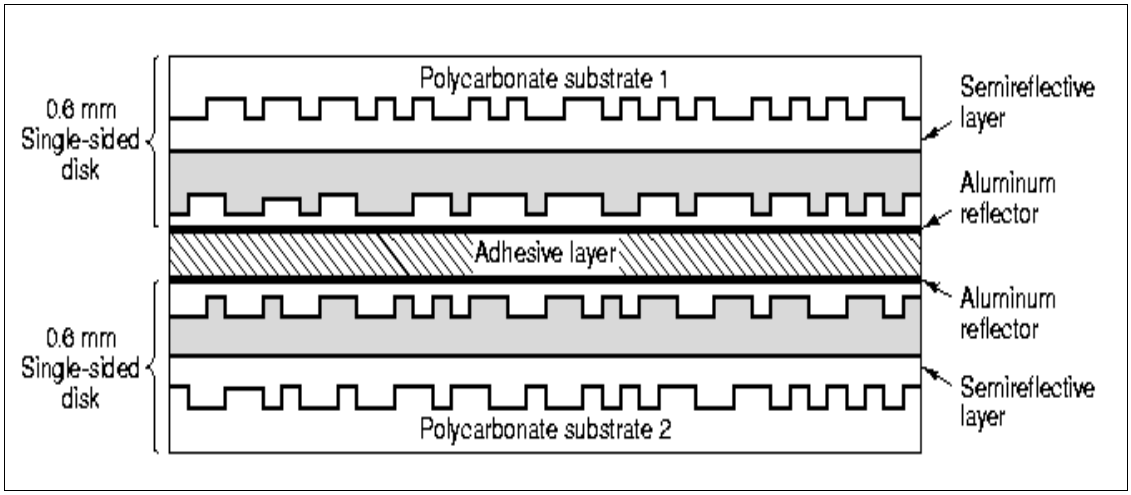


Figure V-36: DVD de doble cara y doble capa

Ejemplo: Especificaciones de la grabadora Samsung TS-H552U

- **Velocidades**: Tiempo de acceso (DVD) 130 ms, Escritura (DVD-R) 16 x (22.160 KB/s), Escritura (DVD-RW) 4 x (5.540 KB/s), Escriturar (DVD+R) 16 x (22.160 KB/s), Escritura (DVD+RW) 4 x (5.540 KB/s), Escritura (DVD+R) Doble capa 2,4 x (3.324 KB/s).
- **Formatos**: DVD-ROM, DVD Video, DVD-R, DVD-RW, DVD+R, DVD+RW, CD-ROM, CD-ROM/XA, CD-R, CD-RW, Audio CD, Video CD, CD Text, Photo CD, CD-Extra, CD-I, Multisesión, Mixto.

En la tabla siguiente mostramos las diferencias físicas entre un CD-ROM y un DVD:

	CD-ROM	DVD
Diámetro	120mm	120mm
Grosor	1.2mm	1.2mm
Ancho de pista	**1.6 micras**	**0.74 micras**
Tamaño mínimo fosos	**0.8 micras**	**0,4 micras**
Longitud onda láser	**780-790 micras**	**635-650 micras**
Capas	**1**	**1 o 2**
Caras	**1**	**1 o 2**
Capacidad por capa	**682**	**4.7 Gb**
Capacidad por cara	**682**	**4.7 a 8.5 Gb**
Total datos	**682**	**17 Gb**
Velocidad (x1)	**1.2 m/s**	**3.49 m/s**
Transferencia (x1)	**153.6 KBps**	**1.385 Mbps**

BLU-RAY y HD DVD

La aparición de la nueva televisión de alta definición (HDTV) hace necesario soportes con una alta capacidad de almacenamiento. Al igual que ha pasado con los DVD+R y los DVD-R, aparecen dos tecnologías diferentes apoyadas por fabricantes distintos que intentan paliar esta nueva necesidad de almacenamiento.

De un lado tenemos la tecnología BLU-RAY (nombre que toma del láser azul-violeta que emplea) apoyado por Apple, Dell, Hitachi, HP, JVC, LG, Mitsubishi, Panasonic, Pioneer, Philips, Samsung, Sharp, Sony, TDK y Thompson. Sus principales especificaciones nos hablan de una capacidad de 25 GB por capa, un láser de 405 nanómetros y unos ratios de transferencia de 36 Mbps. Sin embargo TDK ya está experimentando con prototipos de 100 GB y tasas de transferencia de 72 Mbps.

Por otro lado Toshiba y Nec han apostado por el HD DVD que cuenta con el apoyo del DVD forum y de una gran parte de las principales productoras cinematográficas de Hollywood. Usa el mismo láser azul que su competidora y permite unas capacidades de almacenamiento de 15 Gb por capa. Toshiba a anunciado disco con tres capas que aumentan la capacidad de almacenamiento a 45 Gb.

8. Manejador de disco

El manejar de disco es un software de entrada/salida dependiente del dispositivo cuyas principales funciones son:

- Proceso de cada petición de E/S en bloques.
- Traducción del formato lógico a mandatos del controlador.
- Insertar la petición en la cola del dispositivo, llevando acabo la política de planificación de disco pertinente(FIFO, SJF, SCAN, CSCAN, EDF, etc.).
- Enviar los mandatos al controlador, programando la DMA.
- Bloqueo en espera de la interrupción de E/S.

- Comprobación del estado de la operación cuando llega la interrupción.
- Gestionar los errores, si existen, y resolverlos si es posible.
- Indicación del estado de terminación al nivel superior del sistema de E/S.

8.1. Estructura del manejador de disco

En la figura siguiente muestra la estructura general de un manejador de disco:

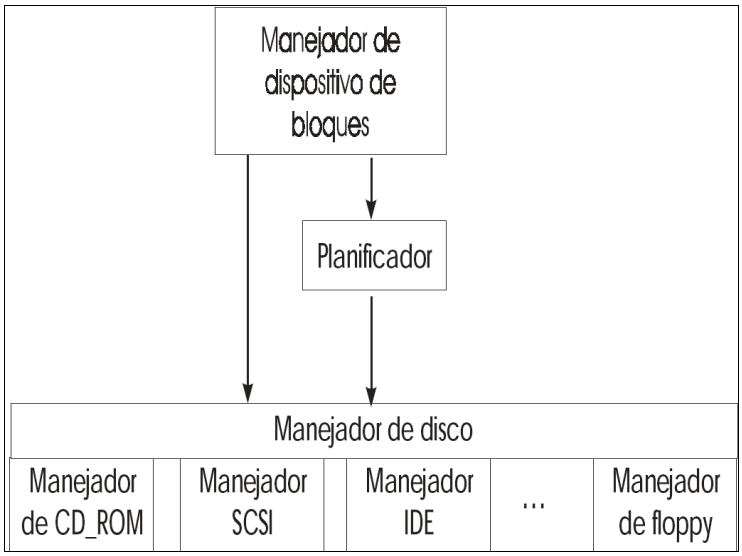


Figure V-37: Estructura general del manejador de disco

Cuando la petición de entrada/salida se produce, el procesamiento de la misma se pasa a un manejar genérico de cada clase de dispositivo. Este se encargará de enviar la petición al manejador del dispositivo concreto o al planificador de disco. Podemos ver al planificador como una cola en donde se van encolando las peticiones que llegan al dispositivo. De entre todas las peticiones que estén en cola se elegirá una que será enviada al manejador del dispositivo.

8.2. Políticas de planificación

En un sistema multiprogramado se realizan muchas peticiones de lectura / escritura de disco. Como vimos el tiempo que se tarda en resolver cada una de estas peticiones consta de dos componentes principales:

- **Búsqueda**: tiempo que tarda el brazo del disco para mover las cabezas hasta el cilindro que contiene el sector deseado.
- **Latencia**: tiempo de espera adicional para que el disco gire hasta ponerse sobre el sector deseado.

Para reducir el tiempo total de acceso es conveniente ordenar las peticiones de forma diferente. Este proceso se conoce como **planificación de disco**. Aunque la mayoría de los algoritmos de planificación del disco se basan en optimizar el tiempo de búsqueda también hay algunos especializados en el tiempo de latencia.

En general cuando se diseña uno de estos algoritmos se persigue que cumpla con las siguientes **características**:

- **Justicia**: deben ser justas con todas las solicitudes
- **Productividad**: lograr una productividad mayor, es decir, atender el mayor número de peticiones por unidad de tiempo.
- **Tiempo promedio de respuesta**: reducir el tiempo promedio de espera (tiempo desde que se realiza la petición hasta que se atiende) más el tiempo promedio de servicio.
- **Igualar la varianza** de tiempos de respuesta: es la diferencia entre los tiempos de respuesta sea lo menor posible (predicibilidad). Que no están unas peticiones esperando mucho tiempo y otras muy poco.

En resumen, se pretende optimizar el tiempo de búsqueda y dar un servicio determinista (de especial importancia para sistemas multimedia y de tiempo real).

Algoritmo	Descripción	Característica
RSS	Planificación aleatoria	Para análisis y simulación.
FIFO	Primero en entrar, primero en salir.	El más justo de todos.
PRI	Prioridad del proceso.	El control se lleva fuera de la gestión de la cola del disco.
LIFO	Último en entrar último en salir.	Maximiza uso de recursos y cercanías.
SSTF	Primero el más corto.	Gran aprovechamiento y colas pequeñas.
SCAN	Recorre el disco de un lado a otro.	Mejor distribución del servicio.
C-SCAN	Recorre el disco en un solo sentido.	Menor variabilidad en el servicio.
SCAN de N-pasos	Scan de N registros a la vez.	Garantía de servicio.
F-SCAN	Scan de N pasos, con N = longitud de la cola al comienzo del ciclo del Scan.	Sensible a la carga.

En los siguientes apartados se presentan diversos algoritmos de planificación. En todos los ejemplos se presupone los siguiente:

- Partimos de la siguiente carga fija para comparar cada uno de los métodos: 98, 183, 37, 122, 14, 124, 65, 67.
- Comenzamos en la pista 53.
- Consideraremos estas referencias como accesos a cilindros o pistas, ya que lo que buscamos es mejorar el acceso del cabezal a los datos.

FCFS (Primero en llegar primero en ser atendido)

Las peticiones se atienden en el mismo orden en el que llegan. Para nuestro ejemplo el resultado de este algoritmo es el siguiente:

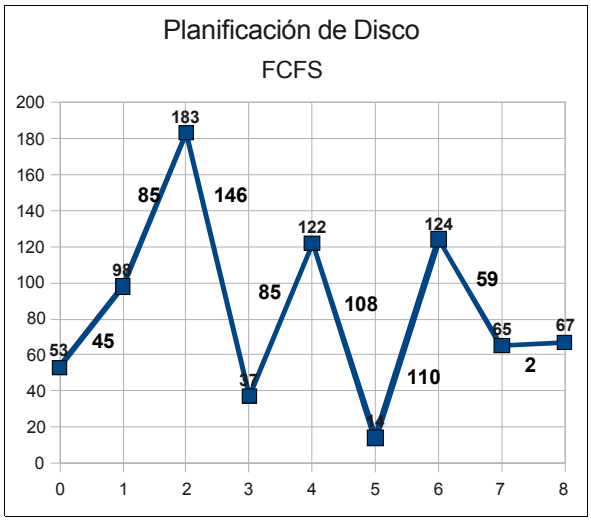


Figure V-38: Planificación FCFS.

Sus principales características son:

- Es justa.
- No intenta optimizar el patrón de búsqueda.
- Es fácil de implementar (no hay sobrecarga).
- La varianza es muy buena.
- El promedio es muy malo (no funciona para situaciones de alta carga).

SSTF (Short Search Time First): Primero el de menor tiempo de búsqueda

Accede a la pista que está más cerca del punto en el que se encuentra el cabezal. Para nuestro ejemplo el resultado de este algoritmo es el siguiente:

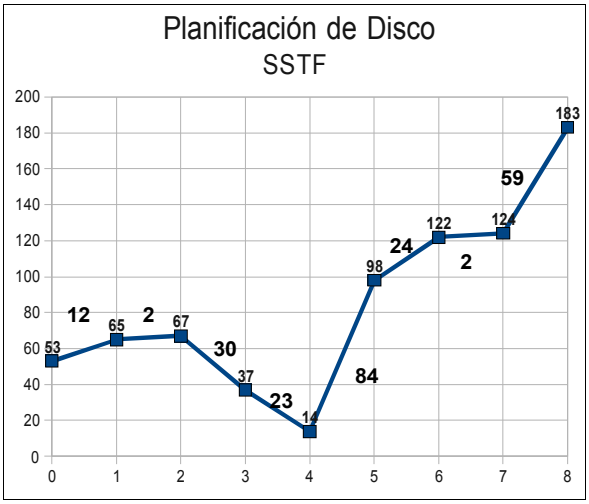


Figure V-39: Planificación SSTF

Sus principales características son:

- No es justa, se ven favorecido los cilindros centrales (postergación infinita).
- Produce mayores tasas de productividad que FCFS.
- Aumenta la varianza de los tiempos de respuesta.
- No es una buena solución para ambientes multiusuario .

SCAN (algoritmo del ascensor)

Atiende aquellas peticiones que tengan menor tiempo de búsqueda, pero en la dirección actual. Para nuestro ejemplo el resultado de este algoritmo es el siguiente:

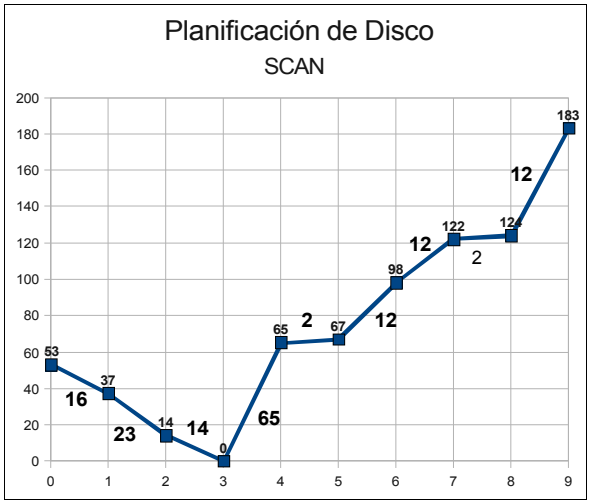


Figure V-40: Planificación SCAN

Sus principales características son:

- Aumenta la productividad.
- Reduce el tiempo de espera respecto al SSTF.

SCAN-N pasos

En esta estrategia, el brazo del disco se mueve de un lado a otro como en SCAN, pero sólo da servicio a aquellas peticiones que se encuentran en espera cuando comienza un recorrido particular. Las peticiones que llegan durante un recorrido son agrupadas y ordenadas para un servicio óptimo durante el recorrido de regreso.

Por tanto en esta estrategia la cola de peticiones se divide en subcolas de tamaño N que se planifican usando un algoritmo SCAN. Si N vale uno este algoritmo se comporta como una FIFO

La SCAN de n-pasos evita la posibilidad de postergación indefinida que tiene lugar si un gran número de peticiones que llegan al cilindro que está siendo servido y guarda estas peticiones para ser servidas durante el recorrido de regreso.

Para nuestro ejemplo el resultado de este algoritmo es el siguiente:

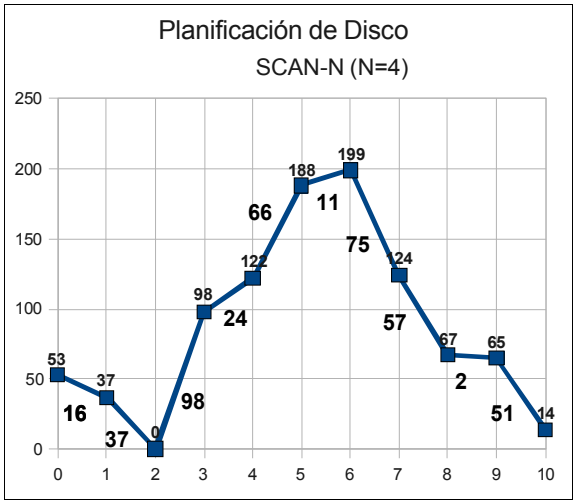


Figure V-41: Planificación SCAN-N (N=3).

Sus principales características son:

- Buena productividad y tiempo de respuesta
- Reduce mucho la varianza si lo comparamos con SSTF y SCAN.

FSCAN

El principal problema que presenta la solución SCAN-N es que el tamaño de los buffers es fijo y no se adapta a las necesidades de carga del sistema. En el algoritmo FSCAN tenemos dos buffer en uno tenemos las peticiones que atenderemos y en el otro las que han ido llegando mientras estábamos planificando el primer buffer. FSCAN se comporta como el algoritmo SCAN-N pero en este caso el valor de N es igual al número de peticiones que estaban en cola al comenzar el barrido en la dirección en curso.

Para nuestro ejemplo el resultado de este algoritmo es el siguiente:

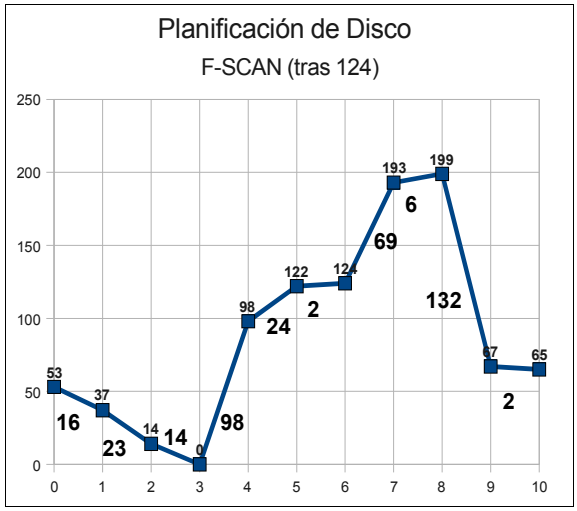


Figure V-42: Planificación F-SCAN

C-SCAN (ascensor acíclico)

Sólo barre el disco en un sentido, cuando termina comienza por el principio, y en el retorno no atiende a ninguna petición.

Para nuestro ejemplo el resultado de este algoritmo es el siguiente:

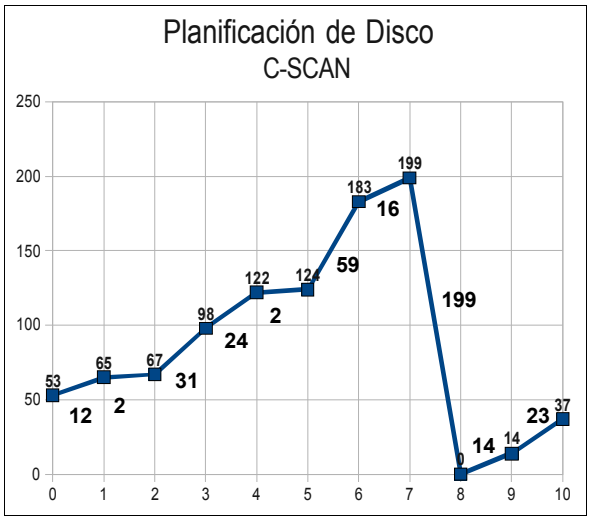


Figure V-43: Planificación C-SCAN

Sus principales características son:

- Varianza pequeña.
- Elimina la discriminación.

LOOK y C-LOOK

En la práctica, ningunos de las variantes de SCAN y C-SCAN se implementan así. Por lo regular, el brazo sólo llega hasta la última solicitud en cada dirección y luego cambia

de dirección inmediatamente, sin primero ir hasta el extremo del disco. Estas versiones de SCAN y C-SCAN se llaman LOOK y C-LOOK, porque miran si hay una solicitud antes de continuar en una dirección dada.

Así pues para nuestro ejemplo la planificación a la que daría lugar el algoritmo LOOK es esta:

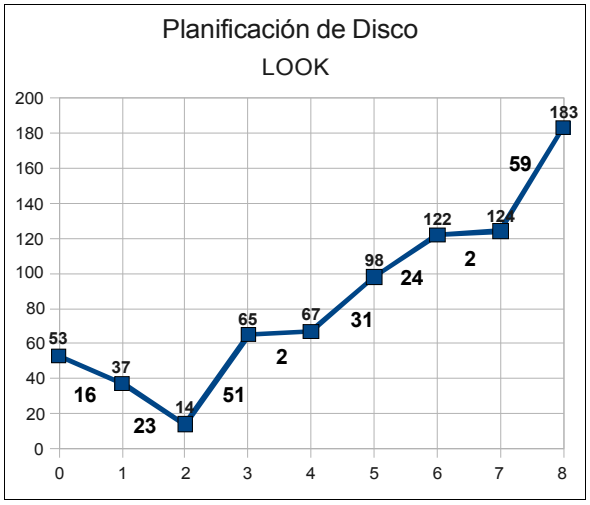


Figure V-44: Planificación LOOK

y la C-LOOK esta otra:

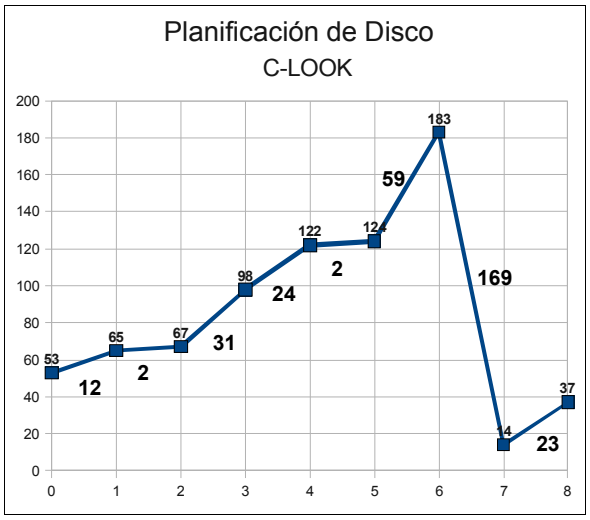


Figure V-45: Planificación C-LOOK

A modo de resumen indicamos las pistas totales recorridas por cada unos de los algoritmos discutidos.

Algoritmo	Pistas recorridas	Algoritmo	Pistas recorridas
FIFO	640	**SSTF**	236
SCAN	236	**LOOK**	208
SCAN-N	437	**F-SCAN**	386
C-SCAN	382	**C-LOOK**	322

Selección del algoritmo de planificación

En general podemos decir que el algoritmo más usado SSTF ya parece el más natural. Ahora bien, para sistemas que usan mucho el disco el SCAN o el C-SCAN, más este último, presentan un mucho mejor rendimiento.

> **Nota**: El rendimiento depende del número y el tipo de peticiones. Las peticiones al disco suelen estar muy relacionadas con la política de asignación de espacio a los ficheros.

Optimización rotacional

Cuando la carga es muy pesada puede ser también interesante reducir el tiempo de latencia. Una de las políticas más utilizadas es la **SLTF** (primero el de menor tiempo de latencia). La optimización de latencia se usa siempre combinada con una política de optimización de búsqueda y bajo cargas fuertes de petición.

9. Fiabilidad y tolerancia a fallos: RAID

Los dispositivos de almacenamiento secundario son bastante propensos a dos tipos de errores:

- **Errores transitorios**: que son debidos a partículas de polvo, fluctuaciones eléctricas, (des)calibración de cabezas, etc y se detectan si al leer el ECC no coincide con el calculado. Para resolver el problema se realizan varias operaciones de entrada salida, si el error persiste se considera que la superficie está dañada y se comunica a niveles inferiores.

- **Errores permanentes:** El tratamiento será diferente dependiendo de si se deben a errores de aplicación (poco que hacer), errores del controlado (tratar de reiniciar el controlador) o errores de superficie: sustituir el bloque por otro de repuesto.

Al sistema de E/S se le exige máxima fiabilidad, ya a que no se desea perder los datos y programas que almacena. Para proporcionar fiabilidad se usan técnicas como los códigos correctores de error o las operaciones fiables. Un claro ejemplo de esto es el uso de los RAID *Redundant Array of Independient Disks* (matriz redundante de discos independientes).

Esta nueva idea surgió en 1998 cuando brecha de velocidad entre la CPU y los discos se había disparado. Se basa en organizar los discos para mejorar su eficiencia y/o su fiabilidad sin necesidad de precisas mejoras en la tecnología de unidad de disco. La idea que subyace en este tipo de configuración es la de tener varios discos que trabajan de forma coordinada con el fin de poder conseguir más rapidez, disponibilidad y/o tolerancia a fallos. Un RAID debe de verse como un SELED (*Single Large Expensive Disk*) a los ojos del sistema operativo.

9.1. Implementaciones

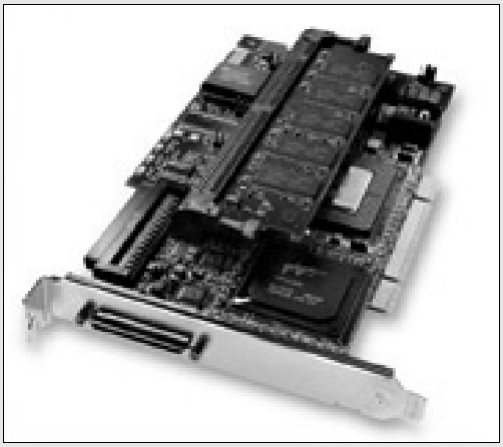
Los sistemas RAID se pueden implementar de dos formas:

- Hardware

- Software

Para la primera necesitamos una controladora RAID (puede estar integrada en la placa base). Las ventajas que tiene esta solución es la velocidad y la eficiencia aunque todo a costa de un coste económico alto.

Ejemplo: Controladora SUPERTRAK-100 IDE PCI 32 BITS ATA-100:



- Descripción: Controladora RAID 0,1,3,5 IDE.
- Accesorios: Incluye chasis para 4 discos que caben en tres bainas de 5.25"
- Capacidad: Discos Ultra ATA/100, Ultra ATA/66 y 33. Hasta 4 discos por controladora
- Características: 4 canales independientes de datos. Procesador RISC Intel i960RD. Memoria de 16 MB ampliable a 128 MB en un SIMM de 72 pins (mín. 4 MB)
- Otras: Detección automática de fallo de disco y reconstrucción transparente. Alarmas sonoras.

Por otro lado la opción de implementar los RAID mediante software requiere que el SO lo gestione. Obviamente es una alternativa mucho más barata y hasta cierto punto más flexible pero necesita soporte del sistema operativo (que puede que no permita implantar todos los niveles RAID) y supone una sobre carga software en el acceso a disco (es más lento).

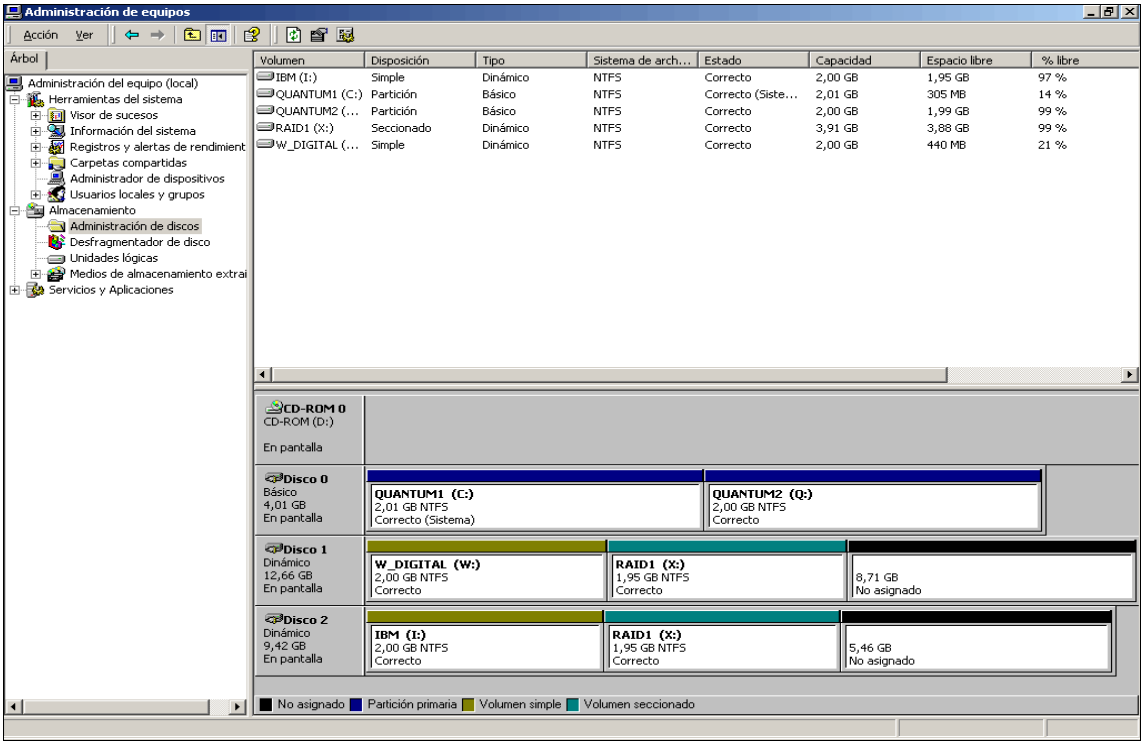


Figure V-46: Raid en windows

9.2. Tipos de RAID

Dispone de varios niveles, u organizaciones, (concretamente 7) para una amplia variedad de modos de almacenamiento de datos. Oficialmente existen siete niveles diferentes de RAID (0-6), definidos y aprobados por el *RAID Advisory Board* (RAB). Luego existen las posibles combinaciones de estos niveles (10, 50, ...). Los niveles RAID más populares. los más usuales:

- RAID 0
- RAID 1
- RAID 0/1
- RAID 5

RAID 0: Disk Striping

Se distribuyen los datos entre distintos discos con el fin de poder acceder a los datos en los diferentes discos en paralelo aumentando así el ancho de banda en la transferencia.

Para logralo los discos se dividen en franjas de sectores y los datos se van almacenando en ellas. A esta manera de distribuir los datos entre varias unidades de disco se denomina **grabación en franjas** (striping).

A este nivel de RAID no se le considera un verdadero raid puesto que no hay redundancia. Como contrapartida obtiene una gran velocidad en las operaciones de lectura y escritura. La velocidad de transferencia de datos aumenta en relación al número de discos que forman el conjunto ya que cada uno de sus elementos está conectado en paralelo respecto al resto.

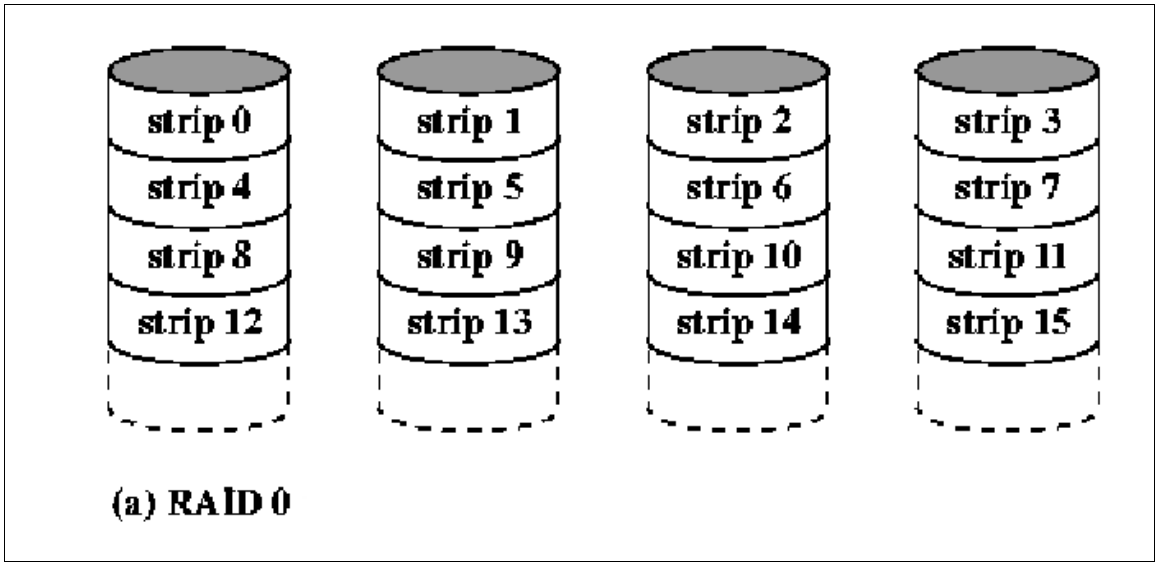


Figure V-47: Raid 0

Las controladoras que soportan RAID 0 suelen soportar una configuración de discos muy similar denominada JBOD (*Just A Bunch Of Disks, sólo un manojo de discos*). No usa *striping*, su funcionamiento se base en crear una unidad lógica formada por diversos discos configurados en JBOD. Las únicas razones que nos pueden llevar a usar esta configuración en detrimento de un RAID 0 son:

- Evita perder tamaño: si tenemos un número de discos de diferentes tamaños, JBOD nos asegura que el total de espacio libre será la suma de todos.
- Recuperación ante desastres: si uno de los discos configurados en JBOD se estropea el sistema completo no se ve inutilizado.

RAID 1 : Mirroring

También llamado “copia en espejo”, realiza de forma automática copia de datos de un disco o varios discos en otros tantos discos con el fin de que si falla uno de esos discos exista otro disco que cuente con exactamente con la misma información.

En los servidores modernos existe la posibilidad incluso de poder reemplazar el disco averiado sin tener que apagar la máquina (“cambio en caliente”).

En la figura siguiente se muestra el funcionamiento de este array:

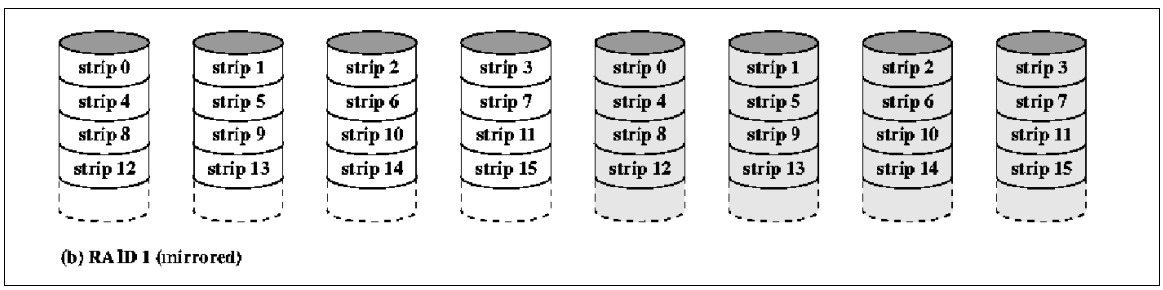


Figure V-48: Raid 1

La redundancia es de 100% pero su costo es grande ya que los datos ocupan el doble de lo estrictamente necesarios. El comportamiento de estos RAID ante lecturas es muy bueno pero no así para escrituras.

RAID 0+1/ RAID 0/1 o RAID 10

Mezcla las características de los dos métodos anteriores para conseguir tanto velocidad como tolerancia a fallos.

En esta configuración los discos de los que disponemos se dividen en dos grupos donde el segundo será una copia exacta del primero (RAID 1). Dentro de cada uno de los grupos la información se almacena exactamente como se hace en un RAID 0.

La principal desventaja que presenta esta arquitectura es su cote ya que requiere un mínimo de cuatro discos (de los cuales sólo dos se usan para almacenar datos útiles) y las unidades se deben añadir en pares cuando se aumenta la capacidad.

RAID 0+1 es una excelente solución para cualquier uso que requiera gran rendimiento y tolerancia a fallos, pero no una gran capacidad.

Este nivel de RAID es el más rápido, el más seguro, pero por contra el más costoso de implementar.

RAID 5

Este array ofrece tolerancia al fallo, pero además, optimiza la capacidad del sistema permitiendo una utilización de hasta el 80% de la capacidad del conjunto de discos. Para lograrlo se distribuyen una serie de bandas de ECC (códigos de detección y corrección) entre todos los discos que componen el RAID (para este sistema, como mínimo 3) con el fin de poder detectar fallos y también, a través de la paridad, corregir algunos de ellos. De esta manera, si cualquiera de las unidades de disco falla, se puede recuperar la información en tiempo real, sobre la marcha, mediante una simple operación de lógica de O exclusivo, sin que el servidor deje de funcionar.

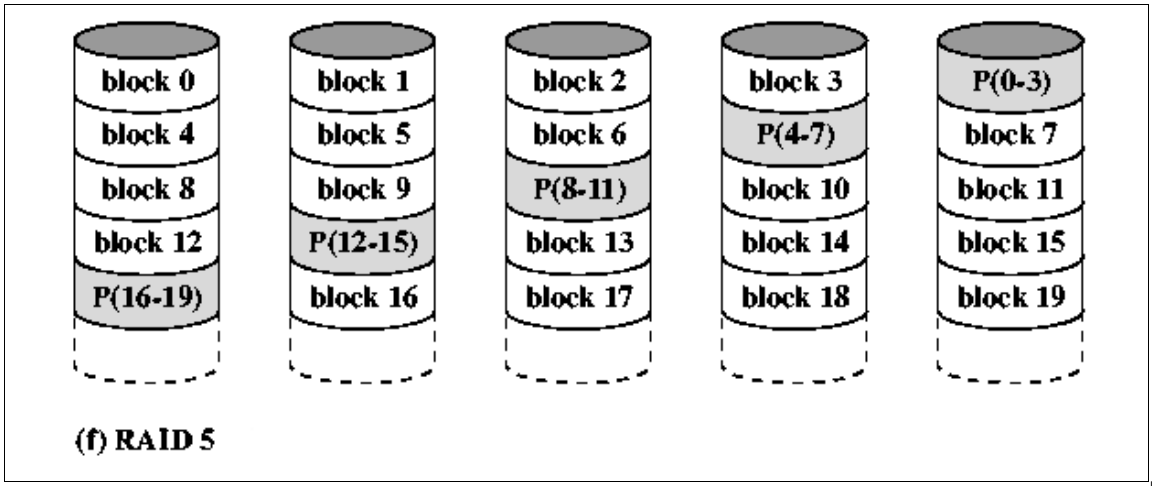


Figure V-49: Raid 5

Al tener las bandas de ECC distribuidas entre los discos duros nos permite evitar el cuello de botella que se produciría si sólo usáramos un disco duro.

Hoy por hoy el RAID 5 es el nivel de RAID más eficaz y el de uso preferente para las aplicaciones de servidor básicas para la empresa. Es el que ofrece la mejor relación rendimiento-coste en un entorno con varias unidades.

Se necesita un mínimo de tres unidades para implementar una solución RAID 5. aunque su resultado óptimo de capacidad se obtiene con siete o más unidades.

9.3. Mejorando estructuras RAID

El uso de tecnologías RAID suelen ir casi siempre unido al uso de otras estructuras que mejoran en gran medida la fiabilidad y disponibilidad del sistema. Algunas de estas características adicionales deseables en todo RAID son:

- Discos “hot spare standby”: discos adicionales instalados en el RAID que entran en funcionamiento cuando algún disco falla.
- Cache protegida con baterías y “espejada”.
- Análisis de la caché y los discos durante periodos de inactividad.
- Respaldo eléctrico para el sistema completo
- Controladores integrados.
- Alertas automáticas (*home-phone capability*).
- Monitorización ambiental.
- Copias instantáneas (*Point-in-Time copy*).
- Copias remotas (*Remote mirroring*).
- Capacidad bajo demanda.

10. Casos de estudio

Para finalizar este capítulo vamos a analizar cómo gestiona la entrada/salida Linux y Windows.

10.1. Gestores de interrupción en Linux

La función del kernel que se ejecuta en respuesta a una interrupción se denomina gestor de interrupción (ISR). Cada dispositivo tiene asociada un ISR que forma parte del *driver* que se usa para gestionar el dispositivo.

Realmente en Linux la gestión de una interrupción se divide en dos partes o *halves*. El ISR forma parte de la primera fase (*top half*) y debe ejecutarse inmediatamente después de recibir la interrupción y sólo realizará aquel trabajo considerado crítico. El resto de tareas se ejecutará en un instante futuro más conveniente, la segunda fase (*bottom half*).

Por lo general las ISR son funciones en C con un prototipo bien definido y suelen ocupar muy pocas líneas para garantizar su rapidez.

Registro de un ISR

Como ya hemos dicho los ISR están dentro del *driver* del dispositivo. Para que estos registren un ISR y la asocien con una línea de interrupción tendrán que invocar a la siguiente función:

```
/* request_irq: allocate a given interrupt line */
int request_irq(unsigned int irq,
                irqreturn_t (*handler)(int, void *, struct pt_regs *),
```

```
                unsigned long irqflags,
                const char *devname,
                void *dev_id)
```

Donde:

- **irq** indica la linea de interrupción a la que vamos a asociar ISR
- **handler** es un puntero a la función que actúa con ISR.
- **irqflags**, puede valer cero o una máscara de una o más de las siguientes banderas:
    - SA_INTERRUPT: inhabilitamos el resto de IRQ para que la ejecución de la ISR sea lo más rápida posible.
    - SA_SAMPLE_RANDOM: la interrupción generada influye en la cola de entropía del kernel cuya misión es la de general números aleatorios.
    - SA_SHIRQ: indica que hay más de un ISR para la IRQ indicada.
- **devname**: es una cadena que representa al dispositivo asociado a la interrupción.
- **dev_id**, se usa cuando una IRQ tiene asociada más de un ISR. Una práctica habitual es poner una referencia a la estructura del dispositivo implementada en el driver ya que es un valor único y suele ser útil para la ISR. Este valor se enviará siempre que se invoque a la ISR.

Si todo ha ido bien la función devolverá un 0 y se creará la entrada correspondiente en /proc/irq. La forma de comprobar si el registro ha ido bien sería:

```
if (request_irq(irqn, my_interrupt, SA_SHIRQ, "my_device", dev)) {
        printk(KERN_ERR "my_device: cannot register IRQ %d\n", irqn);
        return -EIO;
}
```

Al igual que se pueden registrar ISR también se pueden desregistrar gracias a las función free_irq:

```
void free_irq(unsigned int irq, void *dev_id)
```

donde irq es el número de IRQ y dev_id es el identificador que identifica a ISR.

Escribiendo un ISR

La declaración de un ISR es la siguiente:

```
static irqreturn_t intr_handler(int irq, void *dev_id, struct pt_regs
*regs)
```

donde:

- **irq**: es el número de interrupción a la que da servicio.
- **dev_id**: es un puntero a la misma estructura que se pasa cuando se invoca a la función request_irq(). Si esta valor es única actúa como cookie diferenciadora entre múltiples dispositivos que puedan usar el mismo gestor de interrupción.

- **regs**: es un puntero a una estructura que contiene los registros del procesador y el estado del mismo antes de la interrupción

Ejemplo: veamos cómo está implementada la ISR en el driver del RTC (real-time clock). Cuando el driver se carga se invoca a la función rtc_init() que se encarga de inicializar el driver. Entre otras cosas registra el ISR:

```
/* register rtc_interrupt on RTC_IRQ */
if (request_irq(RTC_IRQ, rtc_interrupt, SA_INTERRUPT, "rtc", NULL) {
        printk(KERN_ERR "rtc: cannot register IRQ %d\n", RTC_IRQ);
        return -EIO;
}
```

el código de la funcrión rtc_interrupt, que es el ISR, es el siguiente:

```
/* A very tiny interrupt handler. It runs with SA_INTERRUPT set,
 * but there is a possibility of conflicting with the set_rtc_mmss()
 * call (the rtc irq and the timer irq can easily run at the same
 * time in two different CPUs). So we need to serialize
 * accesses to the chip with the rtc_lock spinlock that each
 * architecture should implement in the timer code.
 * (See ./arch/XXXX/kernel/time.c for the set_rtc_mmss() function.)
 */
static irqreturn_t rtc_interrupt(int irq, void *dev_id, struct
pt_regs *regs)
{
        /*
         * Can be an alarm interrupt, update complete interrupt,
         * or a periodic interrupt. We store the status in the
         * low byte and the number of interrupts received since
         * the last read in the remainder of rtc_irq_data.
         */
        spin_lock (&rtc_lock);

        rtc_irq_data += 0x100;
        rtc_irq_data &= ~0xff;
        rtc_irq_data |= (CMOS_READ(RTC_INTR_FLAGS) & 0xF0);

        if (rtc_status & RTC_TIMER_ON)
            mod_timer(&rtc_irq_timer, jiffies + HZ/rtc_freq +
2*HZ/100);
        spin_unlock(&rtc_task_lock);
        wake_up_interruptible(&rtc_wait);

        kill_fasync (&rtc_async_queue, SIGIO, POLL_IN);
        return IRQ_HANDLED;
}
```

Implementación de la gestión del interrupciones

Aunque la implementación concreta del sistema de gestión de interrupciones depende en gran medida de la arquitectura, en la figura siguiente nos podemos hacer una idea de lo que pasa desde que se genera una interrupción hasta que es atendida por el kernel:

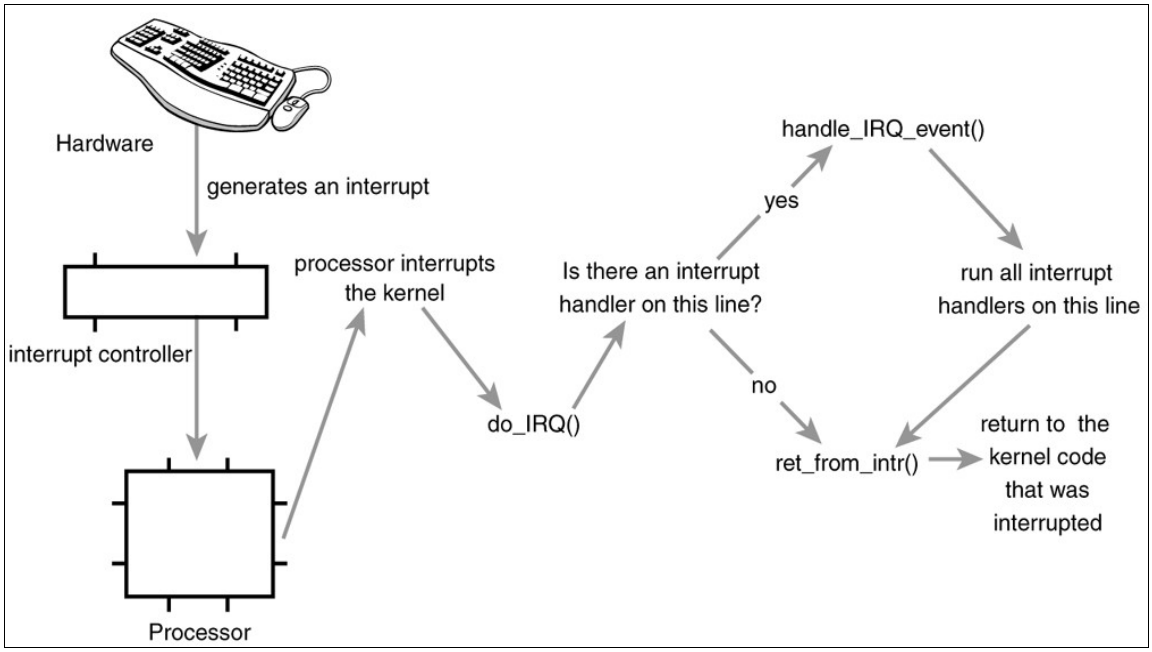


Figure V-50: Pasos para la gestión de una interrupción

Inicialmente un dispositivo genera la interrupción enviando una señal eléctrica sobre el bus del controlador de interrupciones. Si dicha línea esta activa en controlador envía la señal al procesador. El procesador deja lo que está haciendo y pasa a ejecutar el código contenido en un espacio de direcciones pre-establecido. Dicho código forma parte del kernel y en el punto de comienzo del tratamiento software de la interrupción.

Realmente no hay un único punto de entrada, el procesador irá a una posición de memoria u otra dependiendo de la IRQ así el núcleo sabrá qué número de interrupción se ha disparado.

Lo primero que hace el núcleo es almacenar en la pila el número de IRQ y los valores de lo registros (`arch/i386/kernel/entry.S`). Justo después invoca a la función do_IRQ que se encarga de determinar la IRQ que se ha disparado, deshabilitarla y para a ejecutar el ISR adecuado:

```
unsigned int do_IRQ(struct pt_regs regs)
{
      int irq = regs.orig_eax & 0xff; // determina IRQ

      mask_and_ack_8259A(irq); // confirma aceptación y deshabilita IRQ

      // comprueba que hay un manejador válido para la IRQ y lo ejecuta

      handle_IRQ_event(irq, *regs,  struct irqaction *action)

      // limpiamos y volvemos a donde estábamos antes de ejecutar esta
      // función. Si es necesario llama a la función schedule()
      ret_from_intr();

}
```

La ejecución de la ISR la realiza la función `handle_IRQ_event` cuya codificación es similar a la siguiente:

```
asmlinkage int handle_IRQ_event(unsigned int irq, struct pt_regs *regs,
```

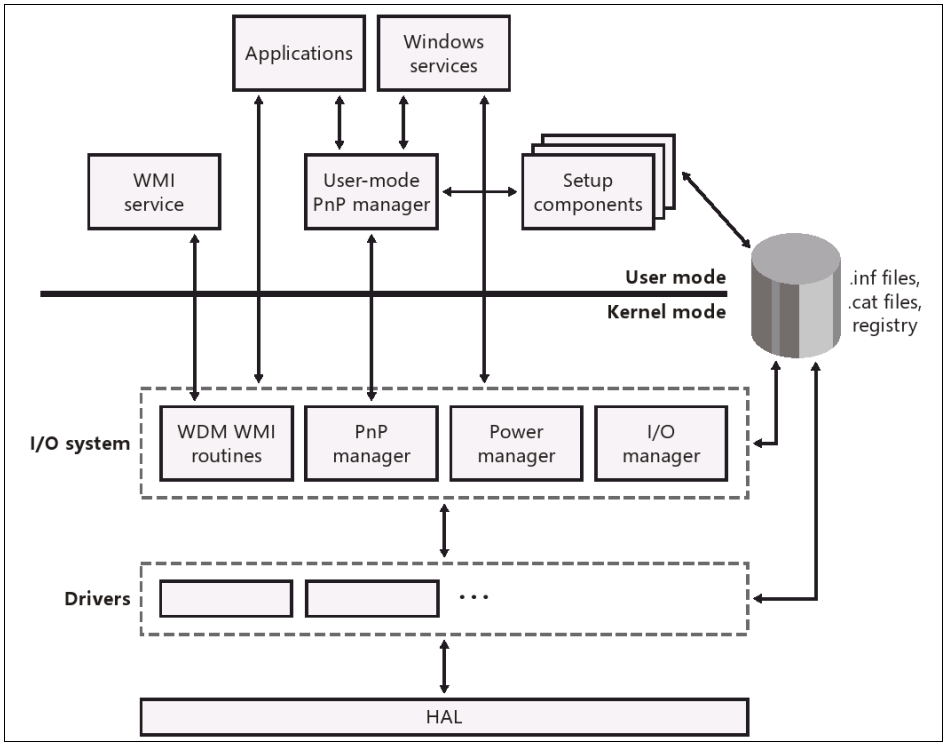
```
                                    struct irqaction *action)
{
      int status = 1;
      int retval = 0;

      //habilitamos interrupciones a no ser que necesitemos exclusividad

      if (!(action->flags & SA_INTERRUPT))
            local_irq_enable();

      // vamos ejecutando los manejadores de la línea, si no está
      // compartida sólo se ejecuta una vez

      do {
            status |= action->flags;
            retval |= action->handler(irq, action->dev_id, regs);
            action = action->next;
      } while (action);

      // ¿contribuimos a la entropía del sistema?

      if (status & SA_SAMPLE_RANDOM)
          add_interrupt_randomness(irq);

      // deshabilidtamos la IRQ

      local_irq_disable();

      return retval;
}
```

10.2. El sistema de entrada/salida de Windows

El sistema de entrada/salida en Windows consiste en diferentes componentes ejecutivos que conjuntamente se encargan de gestionar todos los dispositivos de entrada/salida y proporcionar un interfaz para el acceso a dichos dispositivos

Diseño arquitectónico

Arquitectónicamente el subsistema de entrada/salida de Windows está formado por los componentes que se muestra en la figura siguiente.



Estudiemos un poco más los componentes:

- **El gestor de E/S**: Es el corazón del sistema de entrada/salida y su misión es conectar las aplicaciones y componentes del sistema con los dispositivos lógicos, virtuales y físicos. También es el encargado de definir el API que deben soportar los drivers.

- **Driver de dispositivo**: Un driver proporciona un interfaz de entrada/salida para un tipo particular de dispositivo. Los drivers reciben comandos provenientes del gestor de entrada/salida y que van dirigidos a dispositivos que ellos controlan. Cuando el comando finaliza, los drivers envían el resultado al gestor de entrada/salida. También pueden usar al gestor de entrada/salida para enviar peticiones a otros drivers.

- **Gestor PnP**: Trabajo como un gestor de E/S y es un tipo de driver, denominado driver de bus, ue tiene como objetivo guiar la asignación de recursos hardware así como detectar y responder ante la llegado y/o eliminación de hardware. Cuando un nuevo dispositivos es detectado se encarga de cargar el driver oportuno. Si dicho driver no existe llama al componente ejecutivo Plug and Play que a su vez ejecuta el servicio de instalación de nuevos drivers proporcionado por el gestor PnP en modo usuario.

- **Gestor de energía**: Es el encargado de controlar los cambios de estado de energía de los dispositivos.

- **Windows Management Instruments (WMI):** proporciona rutinas, llamadas proveedor del modelo de driver de Windows (WDM), que permiten a los drivers actuar indirectamente como proveedores usando el proveedor WDM WMI como intermediario de comunicaciones con el servicio WMI que se ejecuta en modo usuario.

- **Ficheros INF:** Son ficheros de instalación de drivers. Son el enlace entre un dispositivo en particular y el driver que asume el control de dicho dispositivo.

Contienen información referente al dispositivo que controlan, el origen y el destino de los ficheros que contienen el driver, las modificaciones que se realizarán en el registro e información de dependencias. La firma digital que certifica que el driver ha pasado los test de compatibilidad de *Windows (Microsoft Windows Hardware Quality Lab* (WHQL)) se almacenan en los ficheros .cat.

- **Capa de abstracción de hardware (HAL):** Proporcionan a los drivers el API necesario para que los drivers puedan controlar al procesador y a las interrupciones sin tener un conocimiento específico de su funcionamiento.

En los siguiente puntos nos vamos a centrar en estudiar los dos primeros elementos.

El gestor de E/S

Cuando en el sistema aparece una operación de entrada/salida no suele involucrar a todos los elementos que forman la infraestructura de entrada/salida. En la imagen 58 vemos el flujo de una operación de entrada/salida.

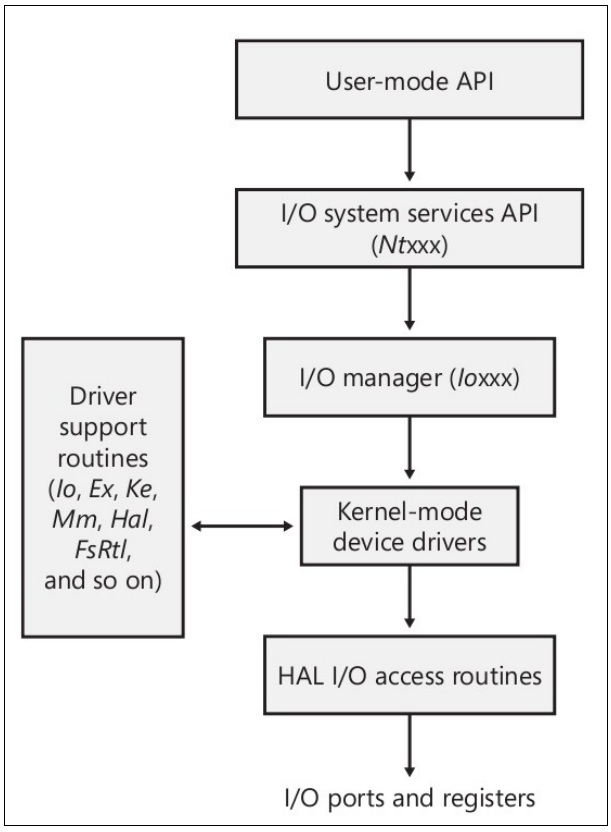


Figure V-51: Roles entre drivers, interacción

Como vemos, cuando se realiza una operación de entrada/salida esta llega al gestor de E/S. Una vez recibida se crea la IRP (*I/O Request Packet*) que definen completamente la petición de entrada/salida. Posteriormente la IRP se pasa al driver que se encarga de gestionar el dispositivo al que va dirigida la petición.

A parte de esto, otras funciones que realiza el gestor de entrada/salida son:

- Proporcionar un código común a todos los drivers con lo que se consiguen drivers más homogéneos y simples.

- Proporcionar un conjunto de servicios de entrada/salida flexibles utilizables por el resto de subsistemas del sistema operativo. Estos servicios van desde operaciones de apertura/escritura simples de dispositivos a operaciones de control de buffers, sincronía, etc.

- Invocar a los driver de dispositivos a través de un interfaz uniforme y modular lo que permite que el gestor de E/S no tenga que tener un especial conocimiento sobre los detalles de implementación del driver.

Drivers en Windows

Un driver no es más que la implementación de un interfaz. El conjunto de rutinas que lo forman (normalmente escritas en C) son invocadas durante todo el proceso que conlleva una operación de entrada/salida.

El interfaz concreto que hay que implementar depende del driver que queremos desarrollar. La clasificación más general de los drivers es la siguiente:

- Drivers que se ejecutan en modo usuario:
    - **Virtual device drivers (VDDs)**: que se usan para emular aplicaciones MS-DOS de 16 bits.
    - **Windows subsystem printer drivers**: que traducen gráficos independientes del dispositivo a comandos para una impresora específica..
- Drivers que se ejecutan en modo kernel:
    - **File system drivers**: aceptan las peticiones de entrada/salida sobre ficheros y satisfacen dichas peticiones generan las suyas propias, solicitudes más concretas para drivers de almacenamiento masivo o de dispositivos de red.
    - **Plug and Play drivers**: trabaja con el hardware y se integra dentro del gestor de subsistema de gestión de energía y PnP. Incluye drivers para dispositivos de almacenamiento, de vídeo, red, etc.
    - **Non–Plug and Play drivers**: también llamados extensiones del kernel ya que aumentan la funcionalidad del sistema proporcionando acceso en modo usuario a servicios y driver que se ejecutan en modo kernel. No se integra con el subsistema de gestión de energía ni de PnP.

Dentro de los drivers que se ejecutan en modo kernel existen diversas clasificaciones atendiendo al tipo de dispositivo que tienen que controlar y al rol que interpretará el driver ante una petición al dispositivo. Con esto último queremos decir que un driver puede interactuar con otros driver tal y como aparece en la figura 60.

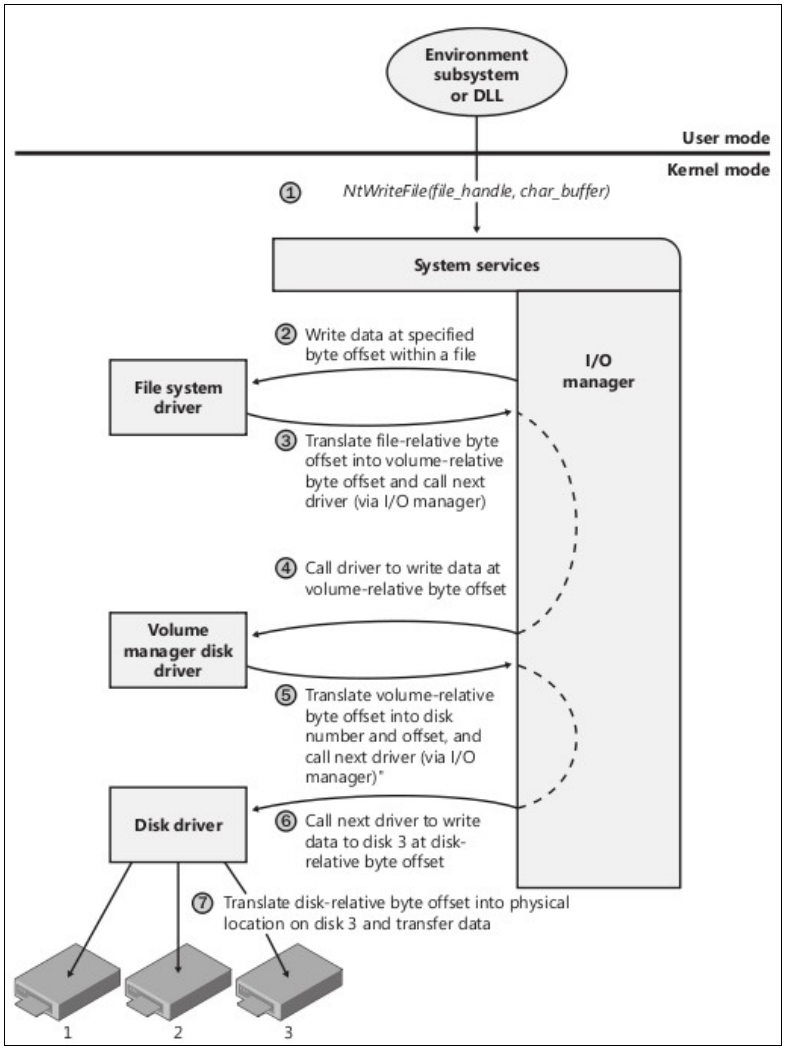
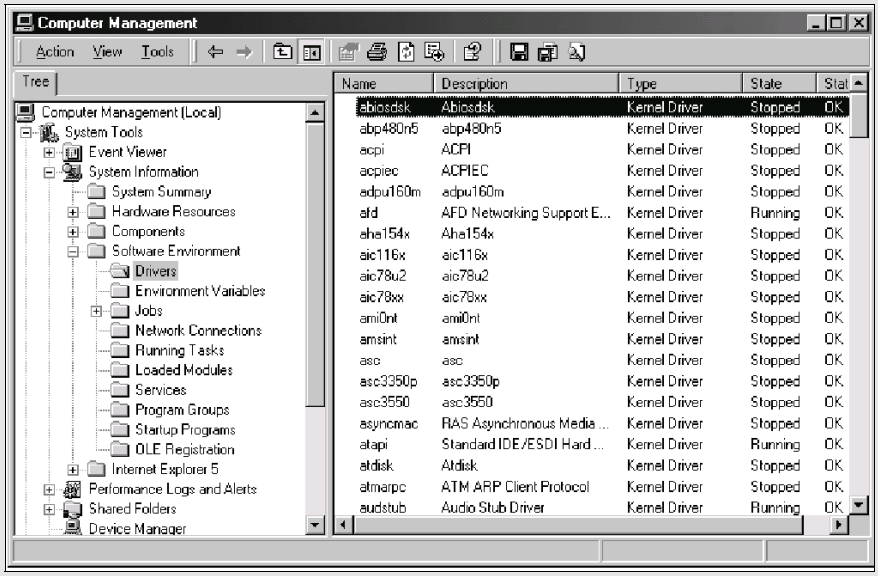


Figure V-52: Roles entre drivers, interacción

Ejemplo: Para consultar los driver que tiene instalado un sistemas Windows 2000 podemos pulsar con el botón derecho sobre el icono "Mi PC" seleccionar la opción de Administración del menú de contexto. Luego tenemos que ir desplegando las siguientes secciones: Herramientas del sistema, Información del Sistema, Entorno Software, Drivers:



En la figura 62 se muestran las principales funciones definidas en el interfaz que deben implementar los drivers:

- **Rutina de inicialización**. Esta rutina se denomina *DriverEntry*. Entre otras cosas se crean los *device objects* que representan a los dispositivos que gestionar el driver invocando a las funciones *IoCreateDevice* o *IoCreateDeviceSecure*. Actualmente todo esto lo realiza la función siguiente. Cuando se crea un objeto dispositivo se le asigna un nombre dentro del directorio \Driver. .

- **Rutina para añadir un nuevo dispositivo**: el sistema *Plug and Play* invoca a esta rutina para indicar al driver que se ha añadido un nuevo dispositivo del cual él es responsable.

- **Rutinas para despachar**: son las principales que un dispositivo debe proporcionar. Ejemplos de estas funciones son: open, close, read, write, close, etc.

- **Comienzo de una entrada/salida**: es una rutina que marca el inicio de una transferencia de datos desde o hacia un dispositivo. Esta función sólo está disponible en aquellos drivers que delegan en el gestor de entrada/salida encolar sus IRP. El gestor de entrada/salida serializa las IRP con lo cual asegura que solo una IRP es procesada a la vez (la mayoría de los drivers procesan más de una).

- **Rutina de servicio de la interrupción (ISR):** en windows el tratamiento de las interrupciones generadas por un dispositivo se divide en dos fases. En una primera fase se ejecuta la ISR en el nivel de prioridad establecido para el dispositivo (DIRQL, *Device Interrupt Request Level*). Para evitar que el sistema se quede bloqueado durante mucho tiempo, la ISR realiza las operaciones imprescindible y encola una llamada diferida a un procedimiento (DPC).

- **Rutina DPC (*Deferred Procedure Call*)**, ejecuta la segunda parte del tratamiento de la interrupción. Realiza la mayoría de las operaciones para atender a la interrupción. La DPC se ejecutan con un nivel de prioridad inferior al DIRQL con lo que conseguimos prevenir el bloqueo innecesario de otras interrupciones.

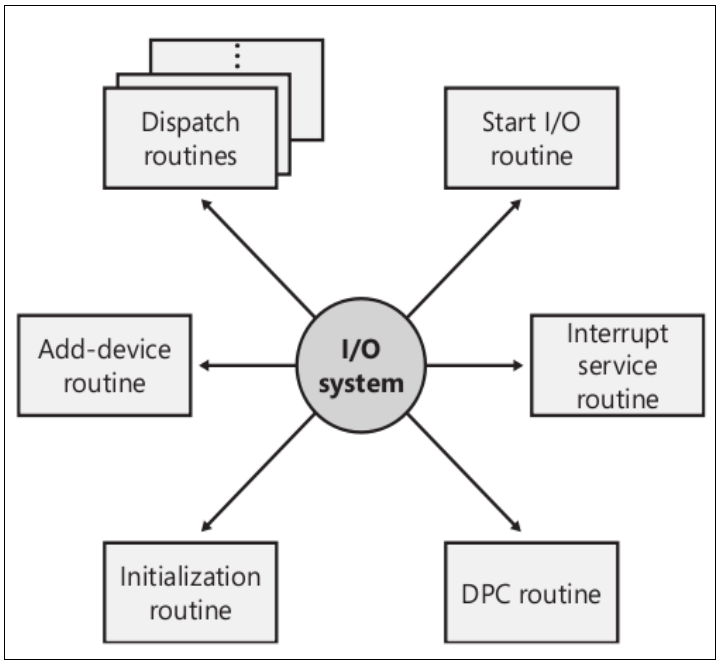


Figure V-53: Rutinas de un driver

Objetos driver y objetos dispositivo

Ya hemos visto que una de las funciones del gestor de entrada/salida es invocar a la una función determinada implementada por el driver que se encarga de la gestión de un dispositivo al que va dirigida una petición, pero ¿Cómo sabe qué driver es? La solución a este problema parte de dos conceptos básicos:

- Un objeto driver representa un driver en el sistema. El gestor de entrada/salida obtiene la dirección de las rutinas de *dispatch* de cada driver (punto de entrada) gracias a los objetos driver.

- Un objeto dispositivo representa a cualquier dispositivo físico o lógico del sistema y describe sus características (como el alineamiento requerido por los buffers y la localización de la cola del dispositivo donde se almacenan las IRPs entrantes).

Cuando un driver se carga en el sistema, el gestor de entrada/salida actúa de la siguiente manera:

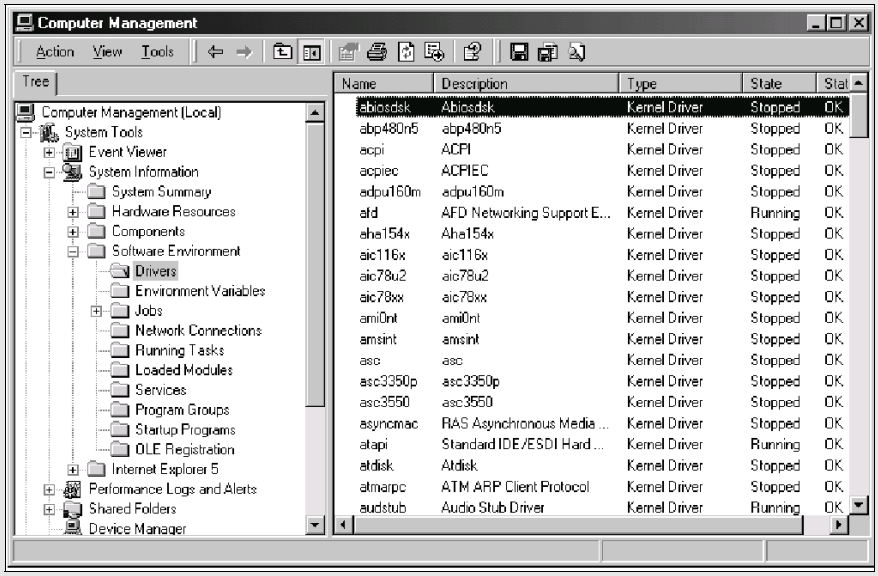
- Crea un objeto del driver.

- Inicializa la instancia invocando al método oportuno. Como resultado de esta inicialización, el driver se crea un objeto que representa al dispositivo que controla.

- Cuando el objeto de dispositivo se crea se le asigna un nombre. Este nombre puede ser generado automáticamente por el gestor de entrada/salida o

definido por el driver. Independientemente de quien lo cree, los drivers se sitúan en la rama \Device dentro del espacio de nombres para la gestión de objetos de Windows. La rama \Device es privada para la aplicaciones por lo que si queremos que uno de estos objetos sea accesible a otras aplicaciones es necesario crear un alias en el directorio \Global.

Ejemplo: Si usamos la herramienta winobj (www.sysinternals.com) podemos ver los nombres de los dispositivos obtendremos algo similar a lo siguiente:



- Cuando el gestor de E/S recibe una petición dirigida a un objeto dispositivo, busca la instancia de dicho dispositivo. Como se puede apreciar en la figura 63 este objeto tiene una referencia al objeto driver que lo gestiona. Una vez obtenida la referencia al objeto driver invoca al método oportuno.

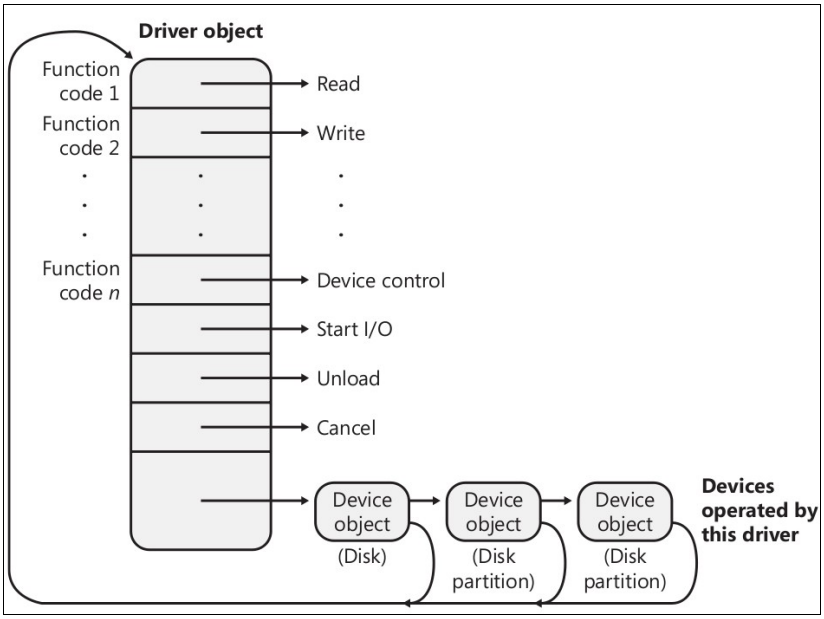


Figure V-54: El objeto driver

- El gestor de entrada/salida libera el espacio ocupado por el objeto driver cuando todas las instancias de los objetos dispositivos que controla han sido borrados.

Ficheros y dispositivos

Un usuario no trabaja directamente con los objetos dispositivos. En su lugar debe usar los objetos fichero.

En la figura 64 se muestran los pasos más importantes que da el sistema cuando se abre un fichero. Como se puede apreciar el sistema le devuelve a la aplicación una referencia (file handle) a un objeto fichero y no a un objeto dispositivo.

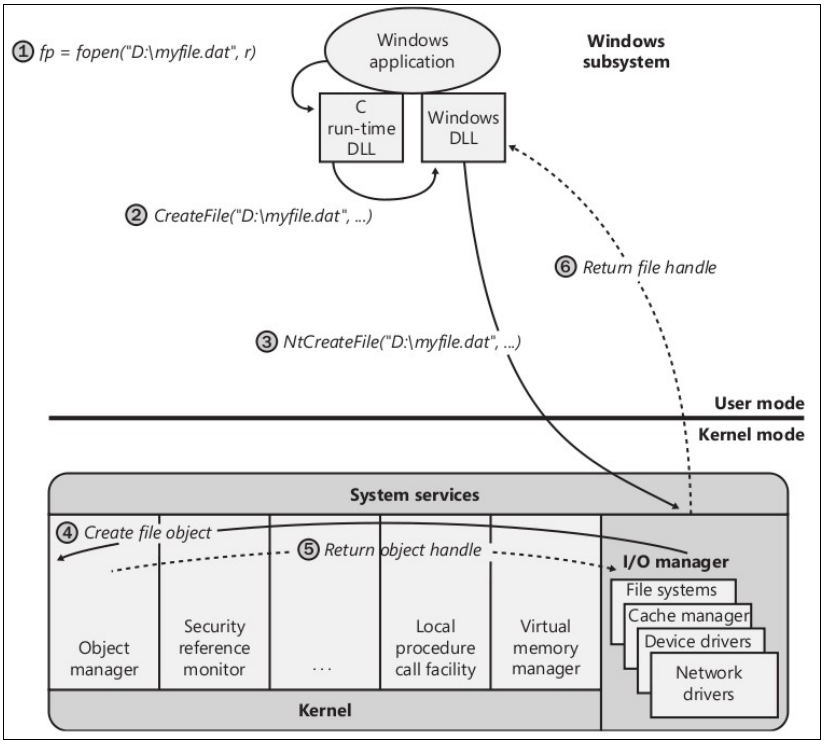


Figure V-55: Abriendo un objeto fichero

Una vez creado el objeto fichero todas las operaciones de entrada/salida que genere la aplicación se realizarán invocando a métodos definidos en el objeto fichero. Ahora bien, estas solicitudes de entrada/salida llegarán al gestor de entrada/salida que se encargará de invocar al método oportuno del driver que gestiona el dispositivo que contiene el fichero. Ya sabemos localizar el driver adecuado si nos dan el dispositivo pero ¿Cómo llegamos a esta información si partimos de un objeto fichero? En la figura 65 vemos los atributos de un objeto fichero como vemos contiene una referencia al objeto dispositivo que contiene al fichero.

Attribute	Purpose
Filename	Identifies the physical file that the file object refers to
Current byte offset	Identifies the current location in the file (valid only for synchronous I/O)
Share modes	Indicate whether other callers can open the file for read, write, or delete operations while the current caller is using it
Open mode flags	Indicate whether I/O will be synchronous or asynchronous, cached or noncached, sequential or random, and so on
Pointer to device object	Indicates the type of device the file resides on
Pointer to the volume parameter block (VPB)	Indicates the volume, or partition, that the file resides on
Pointer to section object pointers	Indicates a root structure that describes a mapped file
Pointer to private cache map	Identifies which parts of the file are cached by the cache manager and where they reside in the cache

Figure V-56: Atributos de un objeto fichero

Ejemplo: Gracias al programa Process Explorer podemos confirmar que todo dispositivos abiertos por un proceso tiene un fichero relacionado: