

Tema 6

Gestión de Entrada/Salida

Sistemas Operativos

Índice

1. **Arquitectura del sistema de E/S** (Tanenbaum [C5.1.1, C5.2 y C5.3])
2. **DISCOS** (Tanenbaum [C5.4.1, C5.4.3 y C5.4.4])
3. **Relojes** (Tanenbaum [C5.5])
4. **Terminales** (Tanenbaum [C5.6 y C5.7])
5. **E/S en Linux** (Tanenbaum [C10.5.1, C10.5.3 y C10.5.4])
6. **E/S en Windows 2000** (Tanenbaum [C11.6.1, C11.6.3 y C11.6.4])

1. Arquitectura del sistema de E/S

- Índice

- 1.1 Objetivos del software de E/S

- 1.2 Dispositivos de bloques y de caracteres

- 1.3 Estructura y componentes del sistema de E/S

- 1.4 Manejadores de interrupciones

- 1.5 Manejadores de dispositivos

- 1.6 Software de E/S independiente del dispositivo

- 1.7 Software de E/S en el espacio de usuario

1.1 Objetivos del software de E/S

1. Independencia de dispositivo

- Se debe poder acceder a los dispositivos de E/S sin tener que especificar previamente de qué tipo de dispositivo se trata

2. Nombres uniformes para los dispositivos

- El nombre de un fichero o dispositivo debe ser simplemente una cadena o un entero, y no depender del dispositivo. Así todos los ficheros y dispositivos se direccionan de un mismo modo: “con un nombre de ruta”

3. Manejo de errores

- Deben manejarse tan cerca del hardware como sea posible
- Subsanan todos los errores posibles
- Comunicando los no subsanables a los procesos de usuario

1.1 Objetivos del software de E/S (ii)

4. Conversión de transferencias asíncronas (controladas por interrupciones) en síncronas (por bloqueo)

- Casi toda E/S física es asíncrona: la CPU inicia la transferencia y realiza otra tarea hasta que llega una interrupción
- Sin embargo, es más sencillo escribir los programas si las operaciones de E/S son **bloqueantes**
- El SO ha de hacer que las operaciones que son controladas por interrupciones parezcan bloqueantes para el programador

5. Compartición de recursos

- Debe permitir la compartición de ciertos dispositivos (como los discos)
- Pero al mismo tiempo debe garantizar el uso exclusivo de otros (como las impresoras)

6. Uso de buffers

- Debe proporcionar almacenamiento temporal en memoria para evitar pérdida de datos (teclado) o acelerar dispositivos (discos)

1.2 Dispositivos de bloques y de caracteres (I)

- Dispositivos de **bloques** (discos)
 - Almacenan información en bloques de tamaño fijo, cada uno con su propia dirección
 - Es posible leer o escribir cada bloque con independencia de todos los demás
 - Dispositivos de **caracteres** (lo que no es disco)
 - Suministra o acepta un flujo de caracteres sin estructurarlos en bloques
 - No es direccionable ni tiene una operación de desplazamiento

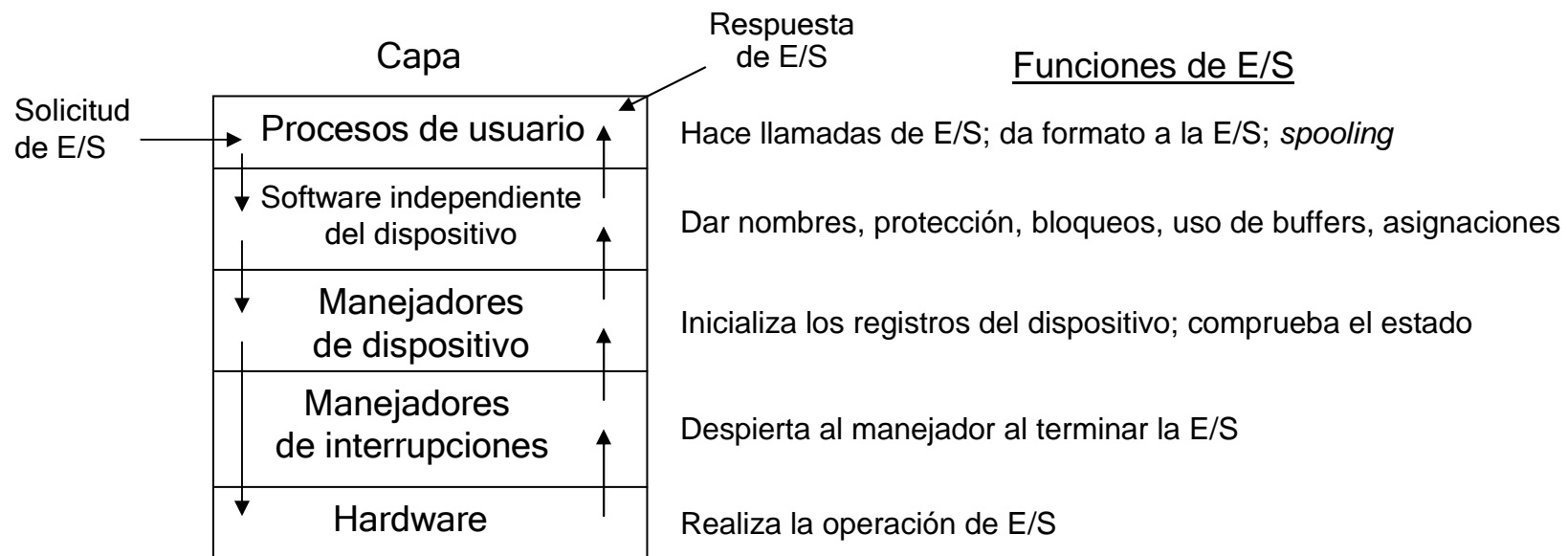
Impresoras, teclados, interfaces de red, etc...
 - Otros, sin embargo, no son direccionables por bloques y tampoco aceptan un flujo de caracteres
- Relojes, pantallas mapeadas a memoria*
- La división es muy útil:
 - Sist. Ficheros \Rightarrow disp. bloques \Rightarrow manejador \Rightarrow disp. Físico
- se ocupa sólo de dispositivos abstractos de bloques

1.2 Dispositivos de bloques y de caracteres (II)

Dispositivo	Tasa de datos
Teclado	10 bytes/s
Ratón	100 bytes/s
Modem de 56k	7 KB/s
Canal telefónico	8 KB/s
Líneas ISDN duales	16 KB/s
Impresora láser	100 KB/s
Escáner	400 KB/s
Ethernet Clásico	1.25 MB/s
USB	1.5 MB/s
Cámara grabadora digital	4 MB/s
Disco IDE	5 MB/s
CD-ROM 40x	6 MB/s
Ethernet rápida	12.5 MB/s
Bus ISA	16.7 MB/s
Disco EIDE (ATA-2)	16.7 MB/s
Interfaz FireWire (IEEE 1394)	50 MB/s
Monitor XGA	60 MB/s
Red SONET OC-12	78 MB/s
Disco SCSI Ultra 2	80 MB/s
Ethernet Gigabit	125 MB/s
Cinta Ultrium	320 MB/s
Bus PCI	528 MB/s
Tarjeta posterior Sun Gigaplane XB	20 GB/s

1.3 Estructura y componentes del software de E/S

- El SW de E/S suele estar organizado en **4 capas**, cada una con una función bien definida y una interfaz clara con las capas adyacentes
 1. Manejadores de interrupciones
 2. Manejadores de dispositivos
 3. Software de E/S independiente del dispositivo (SID)
 4. Software de E/S en el espacio de usuario



1.4 Manejadores de interrupciones (I)

- Se deben ocultar al máximo en el S.O.
- El proceso que solicita la operación se bloquea hasta que termina la E/S
- Cuando se presenta una interrupción el manejador de interrupciones hace lo propio para manejarla. Después avisa al manejador que le solicitó esa operación de E/S
- El manejador despierta al proceso y prosigue su ejecución
- Así, **el efecto real** es que el proceso que antes estaba bloqueado ahora ya puede ejecutarse

1.4 Manejadores de interrupciones (II)

- Incluso a la hora de implementar el SO interesa ocultar las interrupciones (inevitables por molestas que sean)
 - La mejor manera de ocultar las interrupciones es hacer que el controlador que inicia una operación de E/S se bloquee hasta que la E/S haya terminado y se presente la interrupción
 - Dicho controlador puede bloquearse ejecutando *down* en un semáforo, *wait* en una variable de condición, *receive* de un mensaje, etc
 - La subrutina de atención a interrupciones después de hacer su trabajo (si realiza alguno) desbloqueará el controlador que la generó ejecutando *up* sobre semáforo, *signal* variable de condición, enviando un mensaje al controlador bloqueado etc...
 - Este modelo funciona de forma óptima si los controladores se estructuran como procesos del kernel, con sus propios estados, pilas y contadores de programa

1.5 Manejadores de dispositivos

- **Código que depende de los dispositivos**, que sirve para controlarlos
- El fabricante proporciona manejadores de su dispositivo para varios SO
- Cada manejador controla un dispositivo, o una clase de dispositivos
- Forma parte del núcleo del SO, necesita ejecutarse en **modo núcleo** para acceder a los registros de la controladora del dispositivo
- En sistemas Unix antiguos los manejadores se compilaban junto con el kernel (se cambiaba poco de dispositivos y los usuarios estaban familiarizados en compilar el kernel) en lugar de cargarlos de forma dinámica a arrancar o en tiempo de ejecución
- La mayoría de los SO definen una interfaz que todos los manejadores deben soportar (conjunto de procedimientos que el resto del SO puede invocar para pedir a la controladora que realice algún trabajo)
- Al solicitar la orden a realizar, el manejador puede ser que tenga que esperar (bloqueándose hasta que se produzca una interrupción) o que la operación termine sin retraso (el manejador no se bloquea)
- Ha de ser **reentrante** ⇒ durante su ejecución, debe considerar la posibilidad de que se le invoque otra vez antes de terminar

1.6 Software de E/S independiente de dispositivo

- Hay software que no depende directamente del dispositivo
- Dependiente versus independiente: ¿dónde está la frontera?
 - Algunas funciones que podrían efectuarse con independencia del dispositivo en realidad se ejecutan en el manejador por eficiencia o por otros motivos
- Funciones:
 1. Interfaz uniforme del software a nivel de usuario
 2. Asociación de nombres simbólicos de los dispositivos y el manejador correcto:
`/dev/tty0` \Rightarrow nodo-i especial (id. tipo y dispositivo)
 3. Interfaz uniforme para manejadores de dispositivo
 4. Protección de dispositivos \Rightarrow evitar que usuarios no autorizados accedan al dispositivo
 5. Tamaño de bloque independiente del dispositivo
 - Agrupar o dividir sectores para conseguir un tamaño único de bloque lógico
 - Las capas superiores pueden trabajar con dispositivos abstractos

1.6 Software de E/S independiente de dispositivo (ii)

- **Funciones (continúa...)**

6. **Uso de buffers para almacenamiento temporal de los datos**

- Homogeneizan velocidades, se pueden recibir los datos de manera más rápida que la velocidad que pueden salir

7. **Asignación de espacio en los dispositivos de bloques**

- Realizar la gestión de la lista ligada o mapa de bits para administrar el espacio libre es independiente del dispositivo

8. **Arbitrio entre dispositivos de uso exclusivo**

- Examinar las solicitudes de uso de dispositivos y aceptarlas o rechazarlas, según el dispositivo esté disponible o no

9. **Informe de errores**

- Los errores no resueltos por los manejadores deberán ser tratados por el SID, que intentará solucionarlos, y si no puede, deberá informar de los mismos

1.7 Software de E/S en espacio de usuario

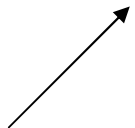
- Procedimientos de biblioteca
 - La mayoría del software de E/S está en el núcleo
 - Sin embargo, también hay procedimientos de biblioteca que se ejecutan en modo usuario y que se encargan de **realizar las llamadas al sistema** (ej. read, printf, etc.)
 - Normalmente preparan el «entorno» adecuado y a continuación realizan la llamada al sistema
- Sistema de spooling
 - Los dispositivos de **uso exclusivo** no se pueden dejar a cargo de programas de usuario (problema: monopolización)
 - El sistema de spooling es una forma de manejar dispositivos dedicados en un sistema con multiprogramación
 - Hay un **demonio** y un directorio de spooling
 - El demonio verifica periódicamente el directorio para saber si hay trabajos pendientes
 - Las impresoras se manejan de esta manera

1.7 Software de E/S en espacio de usuario

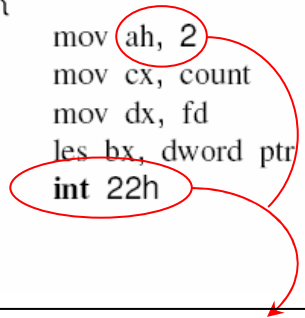
printf en OSO

aplicación

```
int printf(char *fmt, ...)  
{  
    char * pcar;  
    ⋮  
    write(1, pcar++, 1);  
}
```



```
int write(int fd, char far * buffer, int count)  
{  
    asm {  
        mov ah, 2  
        mov cx, count  
        mov dx, fd  
        les bx, dword ptr buffer  
        int 22h  
    }  
}
```



kernel

```
void sai_llamadas(void)  
{  
    salvar_contexto;  
    current->contexto->ax =  
        (tabla_llamadas[current->contexto->ax >> 8].func)();  
    restaurar_contexto;  
}
```

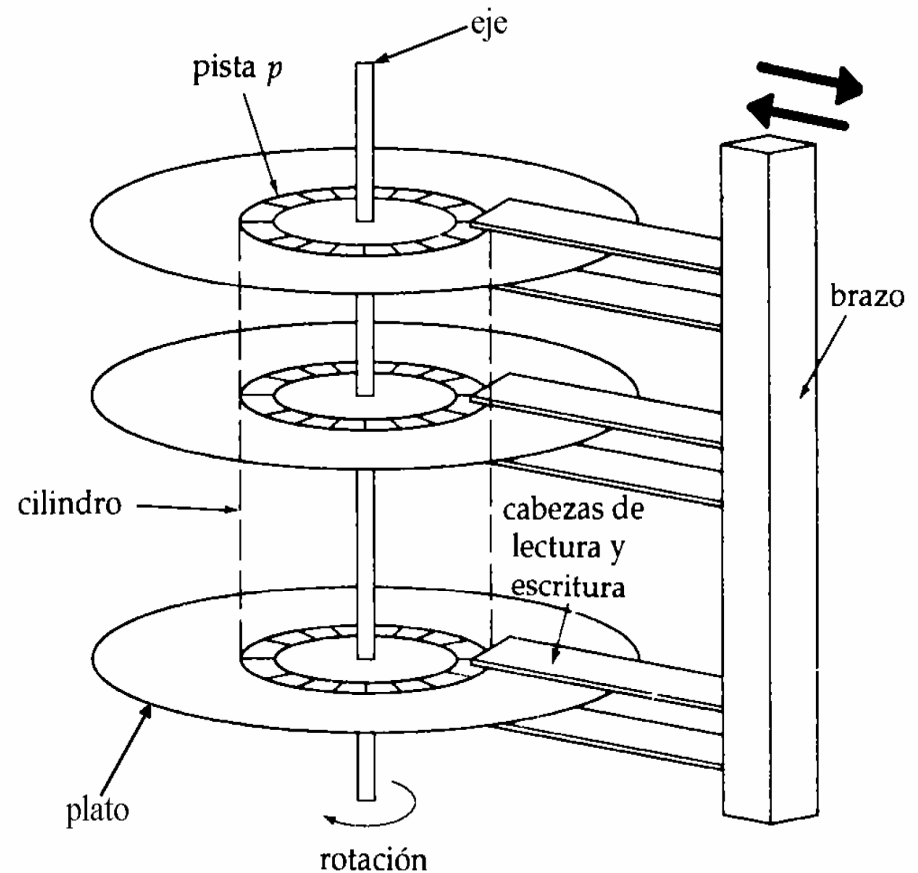
Índice

2. Discos

- 2.1 Hardware para discos. Estructura física
- 2.2 Planificación de los movimientos del brazo del disco
- 2.3 Manejo de errores
- 2.4 Discos en RAM
- 2.5 Fiabilidad y tolerancia a fallos. Sistemas RAID

2.1 Hardware

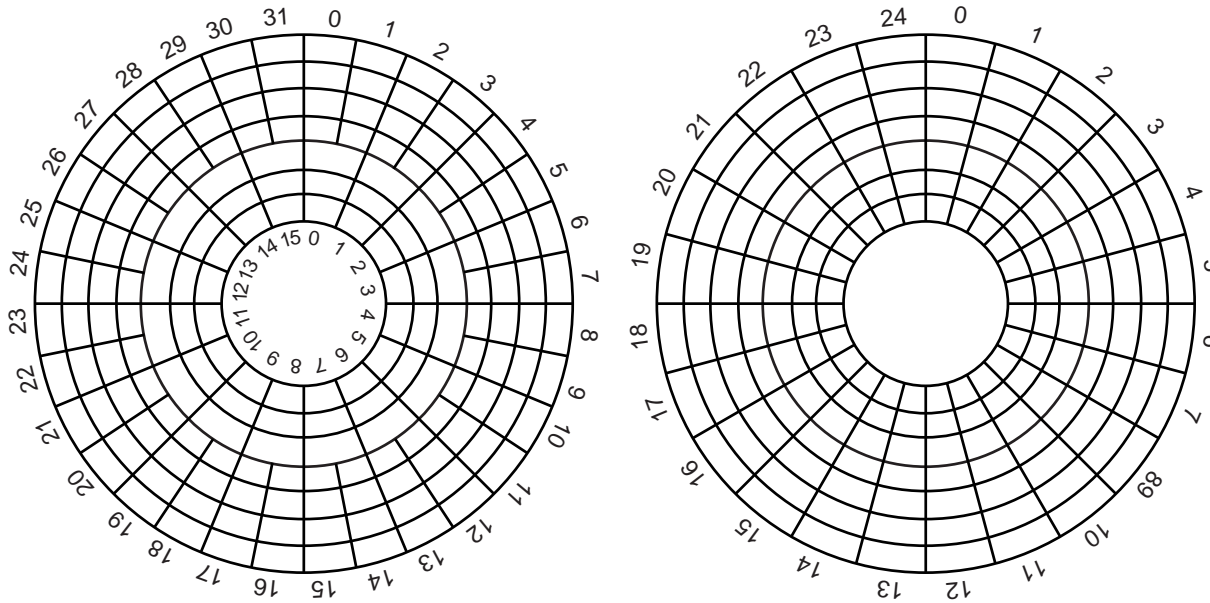
- Serie de platos que giran sobre un eje ($\approx 10.000\text{rpm}$)
- Conjunto de cabezas, una por cada superficie del disco
- Cabezas conectadas a un brazo móvil. Determinan qué pistas conforman un **cilindro**
- Cada cilindro tiene tantas **pistas** como cabezas haya
- Las pistas se dividen en **sectores**
- Las unidades son los **sectores** (bloques físicos) de 512, 1024, 2048 bytes, . .
- Cada sector se identifica como: <cilindro, cabeza, sector>



2.1 Hardware

Parámetro	Disquete IBM de 360 KB	Disco duro WD 18300
Número de cilindros	40	10601
Pistas por cilindro	2	12
Sectores por pista	9	281 (prom.)
Sectores por disco	720	35742000
Bytes por sector	512	512
Capacidad de disco	360 KB	18.3 GB
Tiempo despl. (cilindros adyacentes)	6 ms	0.8 ms
Tiempo despl. (promedio)	77 ms	6.9 ms
Tiempo de rotación	200 ms	8.33 ms
Arranque/paro de motor	250 ms	20 s
Transferencia de un sector	22 ms	17 μ s

2.1 Hardware

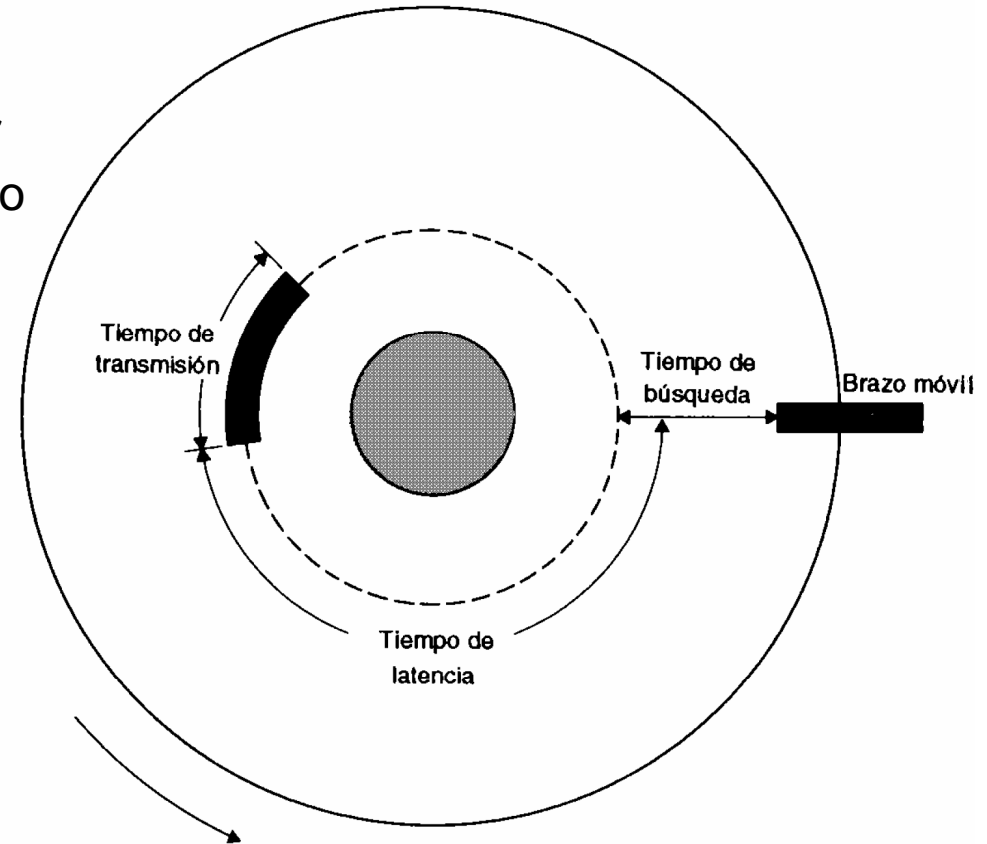


- Geometría real y geometría virtual. La zona exterior tiene 32 sectores por pista y la interior 16 en la geometría real. En la virtual siempre es 25
- Al SO se le presenta la geometría virtual
- Un disco real tiene 16 zonas en vez de 2 y el nº de sectores aumenta 4% de una zona a la siguiente

2.2 Planificación de los movimientos del brazo del disco

- El tiempo de leer o escribir un sector del disco está determinado por cuatro factores:

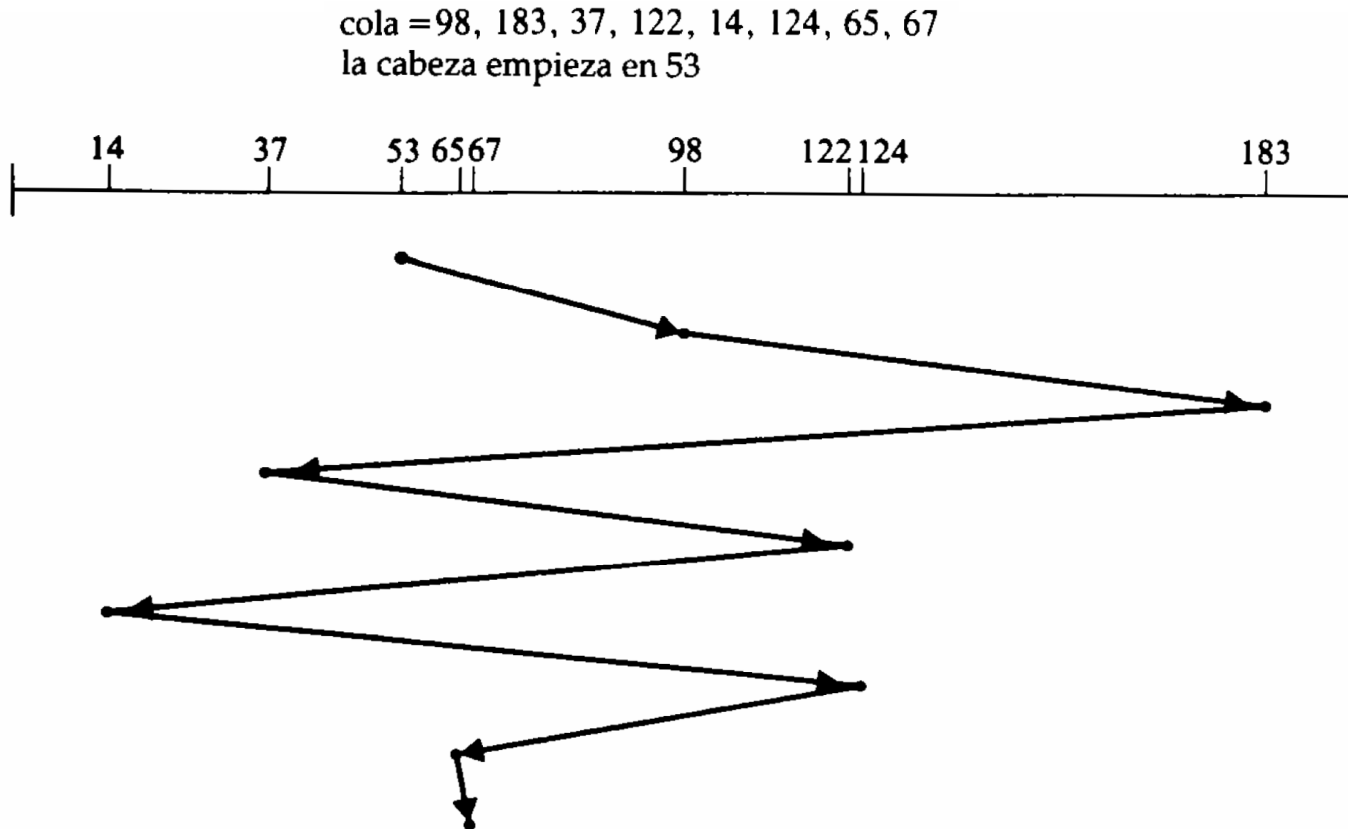
1. Mover el brazo hasta el cilindro \Rightarrow **tiempo de búsqueda**
2. Activar la cabeza adecuada, que supone un tiempo **despreciable**
3. Esperar a que el sector pase ante la cabeza \Rightarrow **tiempo de latencia**
4. Leer o escribir el sector \Rightarrow **tiempo de transmisión**



- El más costoso es el **tiempo de búsqueda** \Rightarrow el objetivo es, por tanto, reducir el **tiempo medio de búsqueda**
- Esto tiene que ser logrado por el **manejador del dispositivo**

2.2 Planificación de los movimientos del brazo del disco (ii)

- Planificación FCFS (*First come, first served*)
 - El manejador acepta solicitudes y las ejecuta en el orden de llegada. No realiza ningún tipo de optimización
 - Problema: tiempos promedio bastante largos

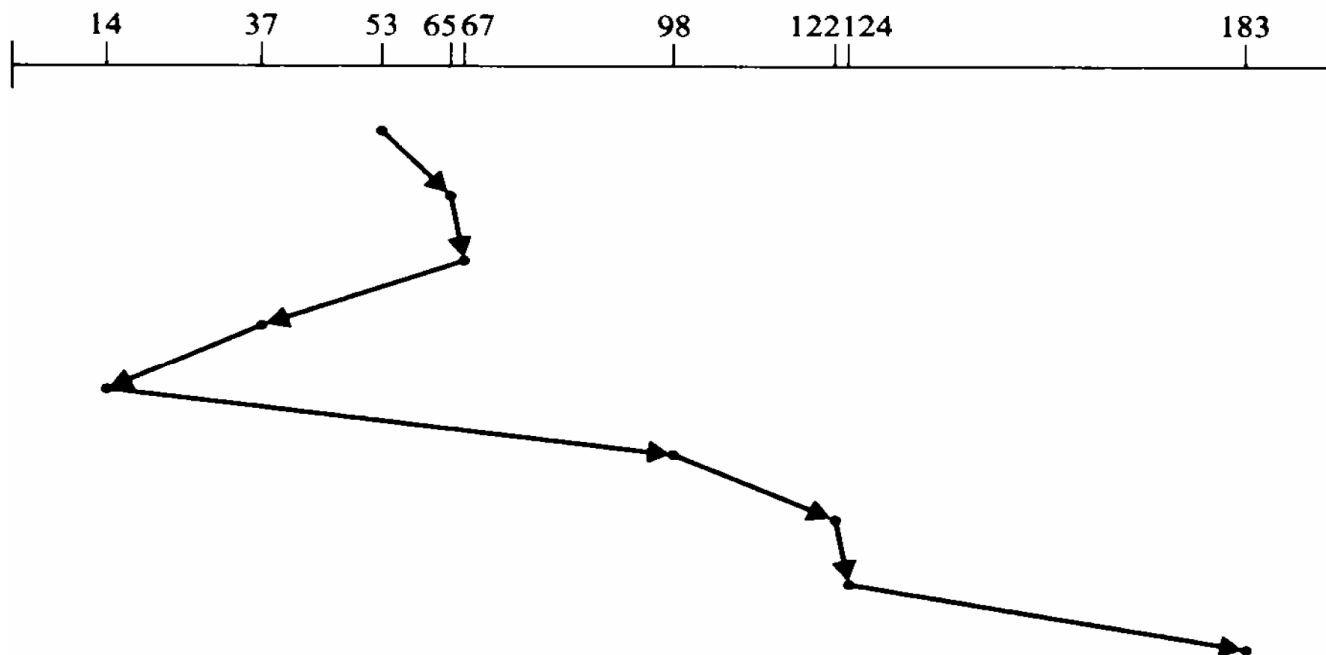


2.2 Planificación de los movimientos del brazo del disco (iii)

- Planificación SSF (*Shortest Seek First*)

- Es demasiado «local» \Rightarrow el brazo tenderá a permanecer en una zona del disco, olvidando las solicitudes de los extremos
- Están en conflicto las metas de reducir al mínimo el tiempo de respuesta y de ser equitativo

cola = 98, 183, 37, 122, 14, 124, 65, 67
la cabeza empieza en 53

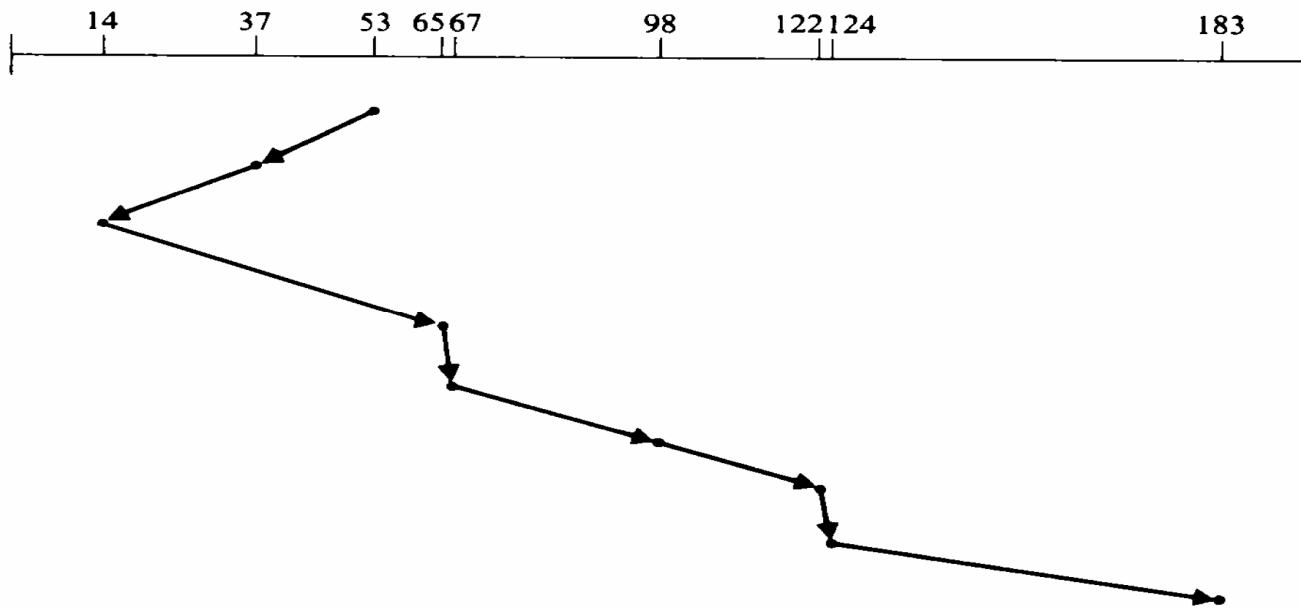


2.2 Planificación de los movimientos del brazo del disco (iv)

● Planificación SCAN o del ascensor

- Siempre avanza en un sentido y resuelve las solicitudes de ese sentido; y cuando no queden más, avanza en el otro sentido
- Se necesita un bit que indique dirección actual
- Tiene cota máxima de cilindros recorridos ($2 * n^0$ de cilindros)

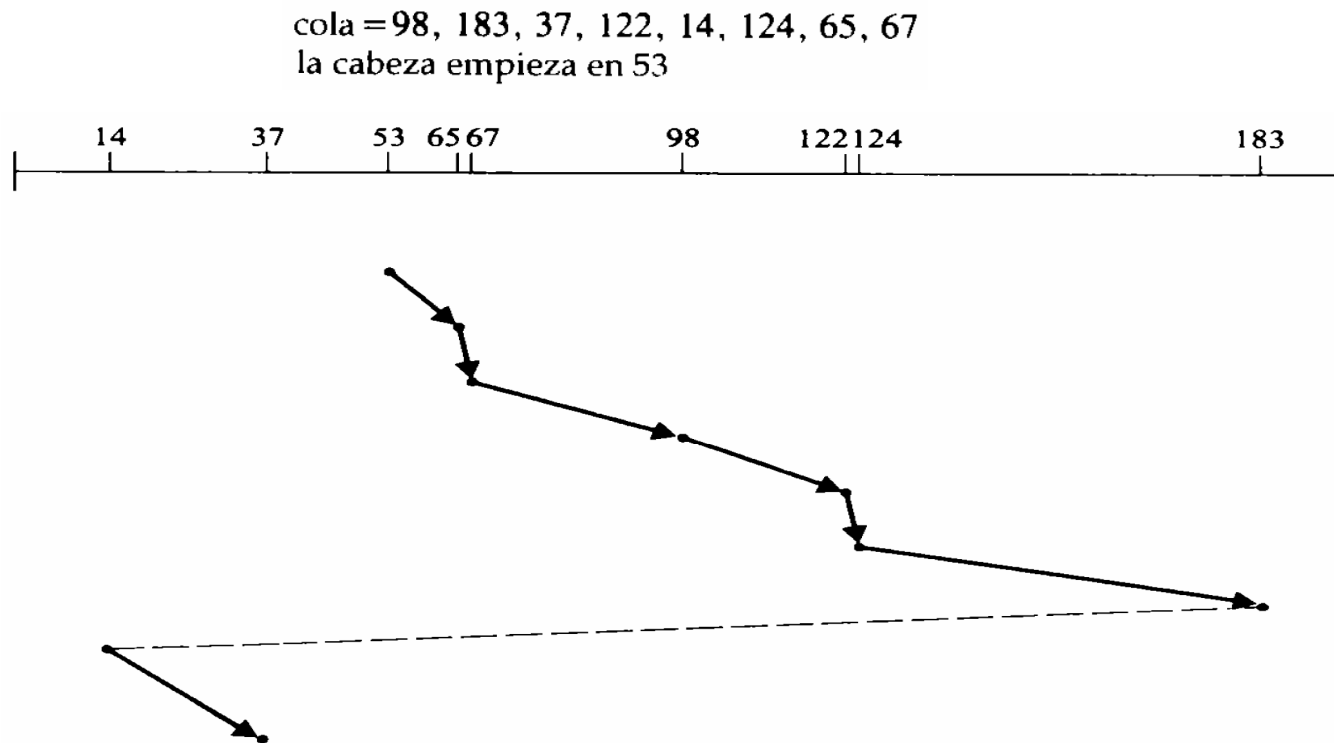
cola = 98, 183, 37, 122, 14, 124, 65, 67
la cabeza empieza en 53



2.2 Planificación de los movimientos del brazo del disco (v)

● Planificación C-SCAN (Circular)

- Igual que la anterior, pero considera al disco como circular. (Al terminar las solicitudes pendientes en un sentido, el brazo vuelve al extremo inicial, y empieza de nuevo a resolver las peticiones en el mismo sentido)
- ¿Por qué? Al volver, habrá menos solicitudes en ese lado



2.2 Planificación de los movimientos del brazo del disco (vi)

- ¿Qué algoritmo elegir?
 - La planificación SSF es bastante habitual. ¿Por qué?
 - Los algoritmos SCAN y C-SCAN son apropiados para sistemas con gran uso del disco. ¿Por qué?
 - Ninguno es óptimo. El rendimiento depende de la cantidad y el tipo de las solicitudes
 - Importante la organización física del Sistema de Ficheros
 - Por ejemplo, en UNIX se podrían poner juntos bloques de directorios y nodos-i (lo que hace FFS)
 - Todos los algoritmos suponen que $t_{busqueda} > t_{latencia}$
 - Si cambia la tecnología \Rightarrow Nuevos algoritmos
- Algunos discos poseen una pequeña caché:
 - El controlador, al leer un sector, lee una pista completa (o casi) y lo coloca en la caché, por si le solicitan sectores de la misma
 - No confundir con la caché del sistema operativo

2.2 Planificación de los movimientos del brazo del disco (vii)

- Ejercicios

- Ejercicio 10 Examen 2º parcial 05/06

- <http://ditec.um.es/so/web-0506/SolucionesTestAEjercicios2Parcial2005-5.pdf>

- Ejercicio 5 Examen febrero 07 (curso 05/06)

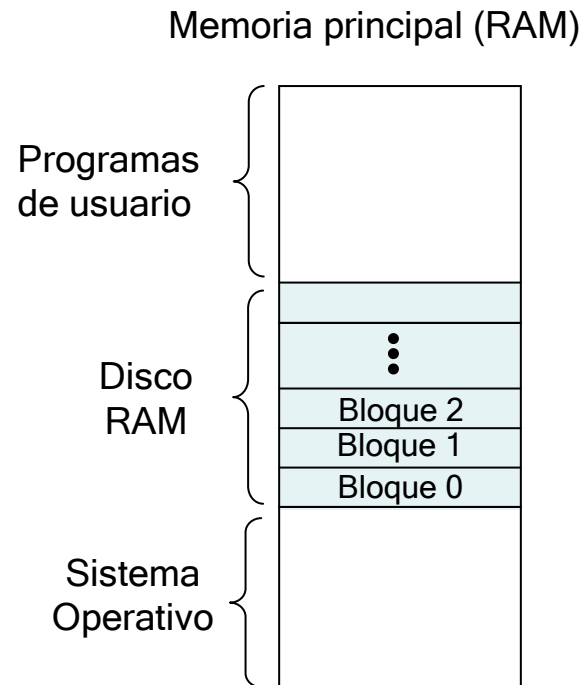
- <http://ditec.um.es/so/web-0506/ProblemasFeb07.pdf>

2.3 Manejo de errores

- Tipos de errores
 - Error de programación
 - Solicitar sector/pista inexistente
 - El controlador verifica los parámetros indicados e informa si son o no válidos
 - El manejador aborta la solicitud que produjo error
 - Error temporal o permanente de CRC
 - Polvo o rotura física del sector
 - Si el error persiste se puede marcar como defectuoso
 - Un controlador “inteligente” puede ocultarlo al manejador (gestionando de los bloques defectuosos)
 - Error de búsqueda
 - Error mecánico del brazo del disco (servo-motor)
 - El controlador comprueba si el cilindro activo es el solicitado, para controlar un posible error
 - O lo soluciona el controlador, o informa del problema al manejador, que normalmente enviará un *recalibrate*
 - Error del controlador
 - ¿Fallo en el hardware? ⇒ Reset ⇒ ¿Arreglado?

2.4 Discos en RAM

- Se usa parte de la memoria principal para almacenar los bloques \Rightarrow acceso instantáneo (no existe retraso rotacional ni búsquedas)
- El disco RAM se divide en n bloques del mismo tamaño que los bloques de los discos reales, salvo que están en memoria
- L/E se calcula la posición del bloque en RAM y se accede a él



2.5 Fiabilidad y tolerancia a fallos. Sistemas RAID

- RAID, *Redundant Array of Inexpensive (Independent) Disks*
 - La brecha entre el desempeño de la CPU y el de los discos se ha ido ensanchando con el paso del tiempo (factor 1000 frente a factor 5 en una década)
 - Combinar varios discos en un «array» de discos, capaces de alcanzar un rendimiento superior a un sólo disco grande y caro (SLED)...
 - Se pretende que varios discos modestos se comporten como un único disco profesional de alto rendimiento
 - Los sistemas RAID se pueden implementar tanto por hardware (mediante un controlador especial) como por software (en el SO)
 - Tienen la propiedad de que los datos se distribuyen entre las unidades de disco, para poder operar en paralelo
 - Hay distintas organizaciones, siendo las más usadas las configuraciones RAID 0, RAID1 y RAID 5, y algunas combinaciones de éstas (como RAID10 = RAID0 sobre RAID's1).

2.5 Fiabilidad y tolerancia a fallos. Sistemas RAID (ii)

■ RAID 0

- El disco virtual único simulado por el RAID se considera dividido en franjas de k sectores cada una
 - Franja 0: sectores 0 a $k - 1$. Franja 1: sectores de k a $2k - 1$
- Las escrituras de franjas consecutivas se reparten entre los discos de manera circular \Rightarrow grabación por franjas
- Las lecturas de franjas consecutivas, cada una del disco correspondiente
- Se consigue E/S paralela sin que el software se entere de ello
- Funciona de manera óptima si las solicitudes son grandes (cuanto más grandes, mejor)
- Tiene un peor funcionamiento cuando las lecturas son sector por sector \Rightarrow los resultados son correctos pero no se consigue paralelismo, por lo que no mejora el desempeño
- No hay redundancia de datos

2.5 Fiabilidad y tolerancia a fallos. Sistemas RAID (iii)

■ RAID 1

- Se duplican los discos, de forma que hay un primario y otro de respaldo
- Al escribir, todas las franjas se escriben dos veces (una en cada disco) \Rightarrow el desempeño de escritura no es mejor que con una sola unidad de disco
- Al leer, se pueden usar cualquiera de las copias, distribuyendo la carga entre los discos \Rightarrow el desempeño de lectura puede ser dos veces mejor
- Tiene una buena tolerancia a fallos \Rightarrow si una unidad no funciona, se usa la otra copia
- Para realizar la recuperación \Rightarrow instalar el nuevo disco y copiar en él todo el contenido del otro

2.5 Fiabilidad y tolerancia a fallos. Sistemas RAID (iv)

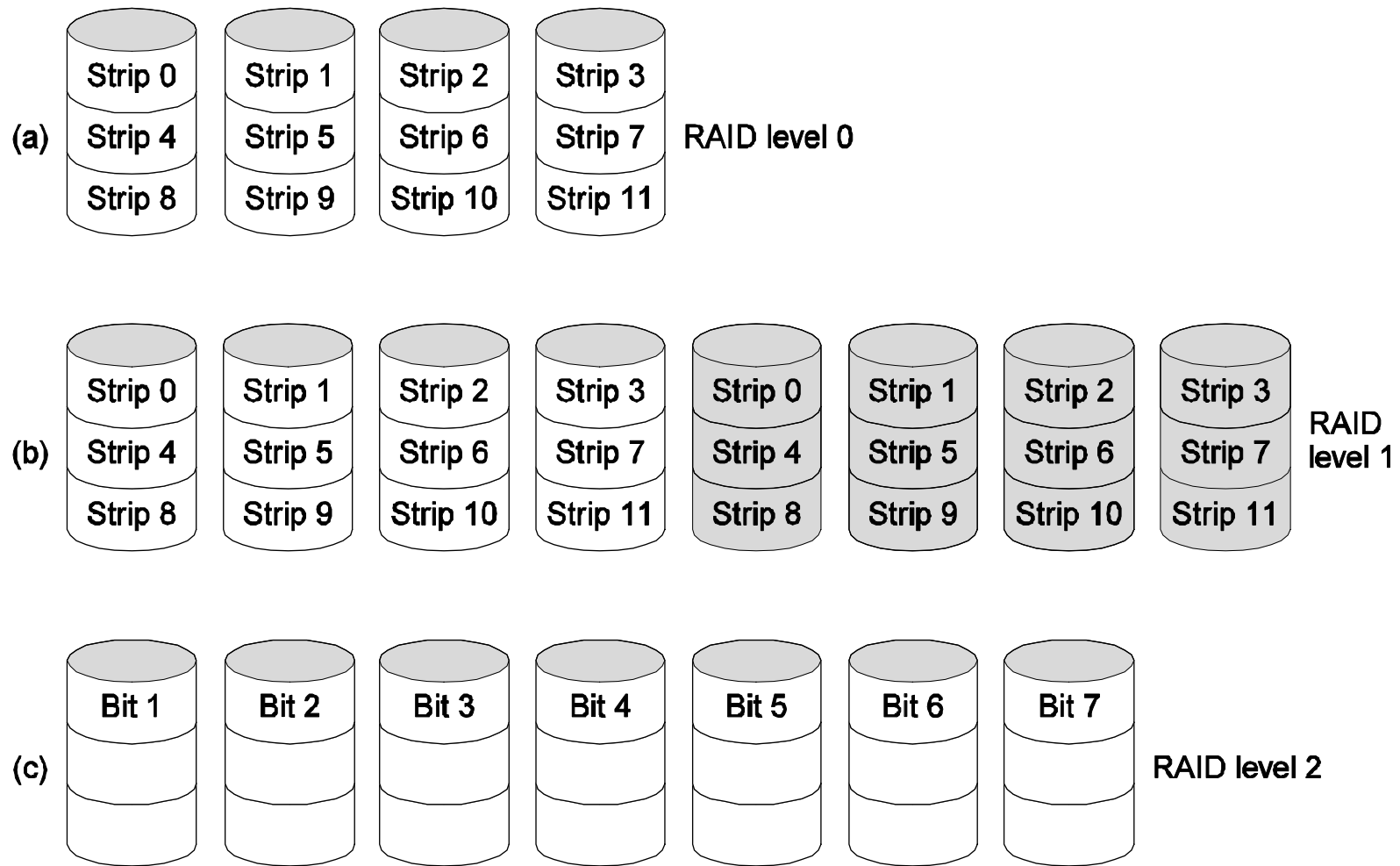
- RAID 4

- Trabaja, de nuevo, con franjas de sectores
- En una de las unidades de disco se calcula la paridad de las otras franjas
- Protege contra la pérdida de una unidad de disco
- Tiene desempeño bajo para actualizaciones pequeñas \Rightarrow la modificación de un sector supone el cálculo de paridad (o leer las otras franjas para recalcular, o con los datos antiguos y la paridad antigua recalcular la nueva)
- La unidad de disco que guarda la paridad se convierte en un cuello de botella

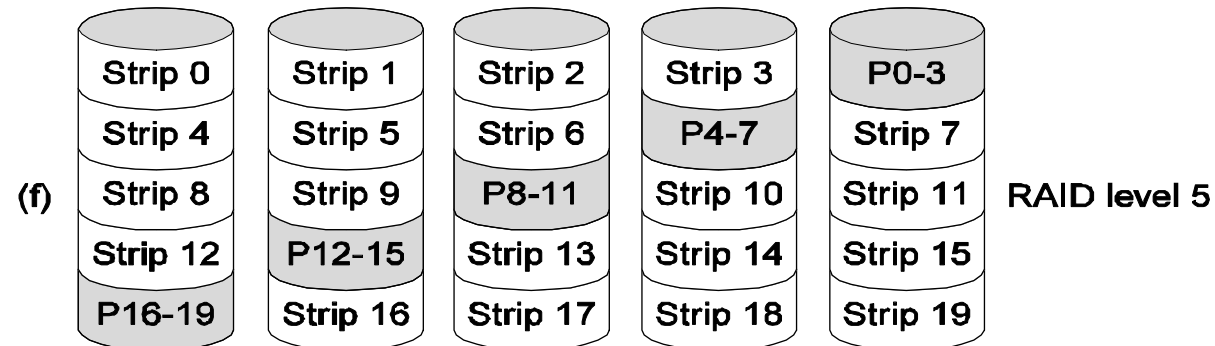
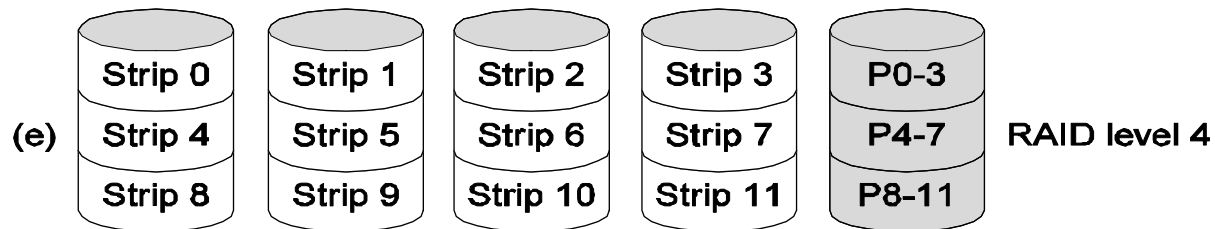
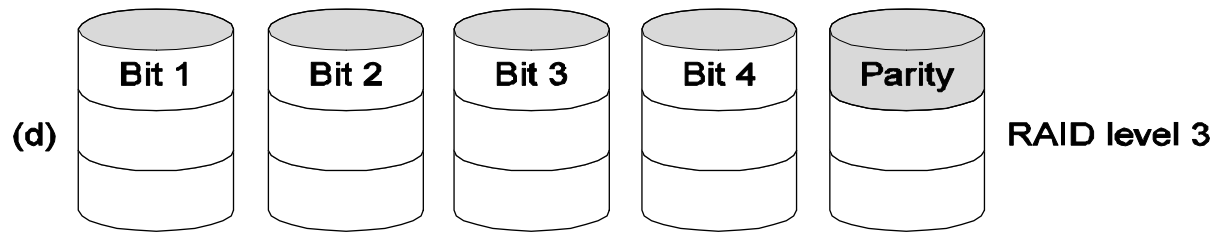
- RAID 5

- Idem a RAID 4, pero distribuye la franja de paridad de manera uniforme entre todas las unidades

2.5 Fiabilidad y tolerancia a fallos. Sistemas RAID (v)



2.5 Fiabilidad y tolerancia a fallos. Sistemas RAID (vi)



Índice

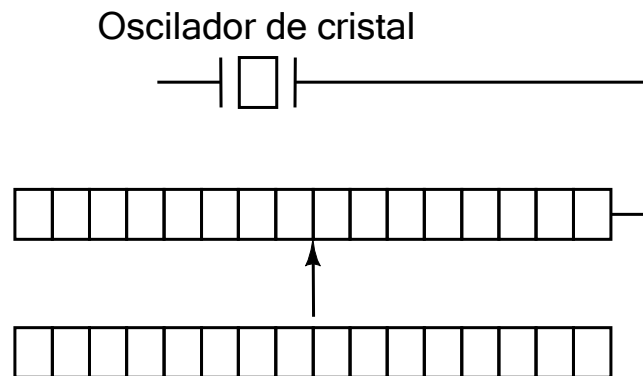
3. Relojes

3.1 Hardware para relojes

3.2 Software para relojes

3.1 Hardware para relojes

- Dos tipos de relojes:
 - Conectados a una línea de alimentación eléctrica de 110 a 220 volts. y causan una interrupción en cada ciclo de voltaje, a 50 o 60Hz. Son poco comunes
 - **Relojes programables**, formados por tres componentes: un **oscilador de cristal de cuarzo**, un **contador** y un **registro**. Generan una señal periódica de gran precisión



El contador se decrementa en cada pulso

El registro se utiliza para cargar el contador

3.1 Hardware para relojes (ii)

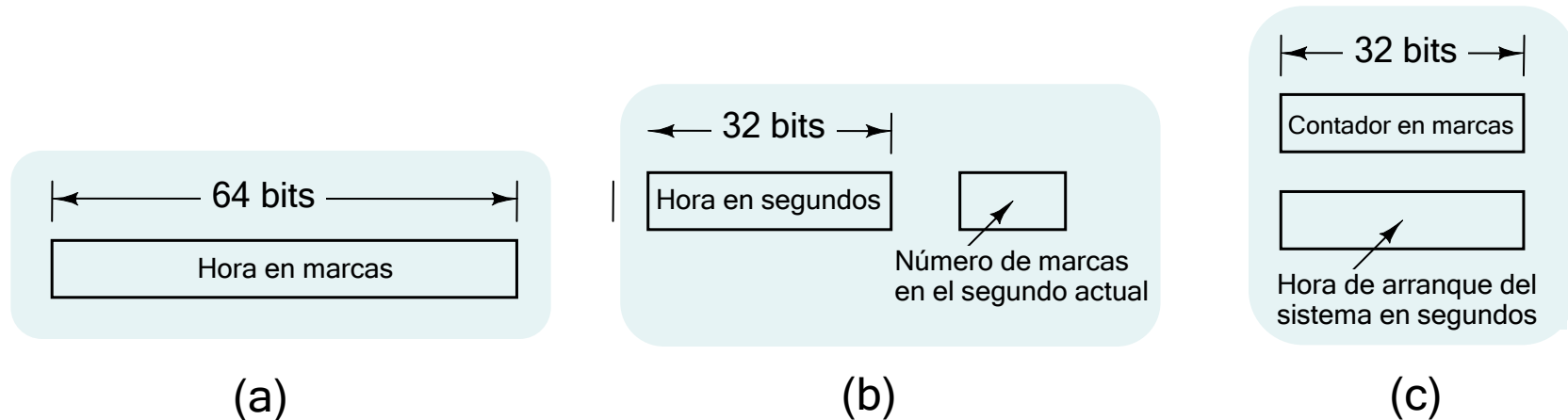
- Modos de operación habituales en un reloj programable:
 - **Modo de disparo único:** al poner en marcha el reloj se copia el valor del registro en el contador. En cada pulso el contador disminuye. Cuando el contador llega a 0, se produce una interrupción, y se detiene hasta que el software lo vuelve a iniciar de forma explícita
 - **Modo de onda cuadrada:** cuando el contador llega a 0 y se provoca una interrupción, el registro se copia de forma automática en el contador, y todo el proceso se repite otra vez
 - Estas interrupciones periódicas se llaman **tics de reloj**
- Funciones principales:
 - Registran la hora del día
 - Evitan que un proceso monopolice la CPU
- Casi todos los ordenadores tienen además un reloj de respaldo (tipo reloj digital de pulsera) alimentado por batería para evitar que se pierda la hora cuando el ordenador permanece apagado

3.2 Software para relojes

- El hardware sólo genera interrupciones a intervalos conocidos
- Todo el manejo del tiempo se realiza en software (**manejador del reloj**). Las tareas de este software son
 - Mantener la hora del día
 - Evitar que los procesos tengan la CPU más de lo debido
 - Contabilizar el uso de la CPU
 - Controlar las alarmas para los procesos de usuario que lo solicitan (llamada al sistema *alarm*)
 - Implementar cronómetros guardianes para el propio sistema (*watchdogs*)
 - Ayudar a realizar resúmenes, monitorización, estadísticas, . . .
- Debe realizar el proceso con **rapidez** (ya que se ejecutará varias veces por segundo, en cada tic de reloj)

3.2 Software para relojes (ii)

- Controlar la hora del día
 - Sólo requiere incrementar un contador en cada tic de reloj
 - Se deben contar “marcas” desde un punto de referencia
 - (UNIX: 0:00, 1/1/1970). Tres ejemplos:
 - Contador de 64 bits que cuenta marcas: 60Hz $\Rightarrow \approx 9.8 \cdot 10^9$ años
 - Contador de 32 bits que cuenta segundos + contador auxiliar de marcas (hasta 1 seg.): ≈ 136 años \Rightarrow fin 2106
 - Contador + hora inicial del sistema: cuenta tics desde el momento en que se puso en marcha el sistema



3.2 Software para relojes (iii)

- Control del tiempo de ejecución de procesos
 - Al iniciar un proceso, el planificador asigna a un contador el valor del quantum de ese proceso en tics de reloj
 - En cada interrupción del reloj \Rightarrow el contador del quantum disminuye en 1
 - Contador con el valor del quantum = 0 \Rightarrow se llama al planificador
- Contabilizar el uso de CPU
 - Iniciar un segundo reloj hardware cada vez que se inicia un proceso (más precisa y más costosa)
 - Al parar el proceso, se determina durante cuánto tiempo se ejecutó
 - Al producirse interrupciones este reloj se podría parar
 - Asignar marcas al proceso actual (menos precisa ¿por qué?)
 - Una variable global apunta a la entrada de la tabla del proceso en ejecución
 - En cada tic de reloj se incrementa la variable en 1
 - Se contabiliza un tic entero sin tener en cuenta si el proceso llegó a consumirlo por completo

3.2 Software para relojes (iv)

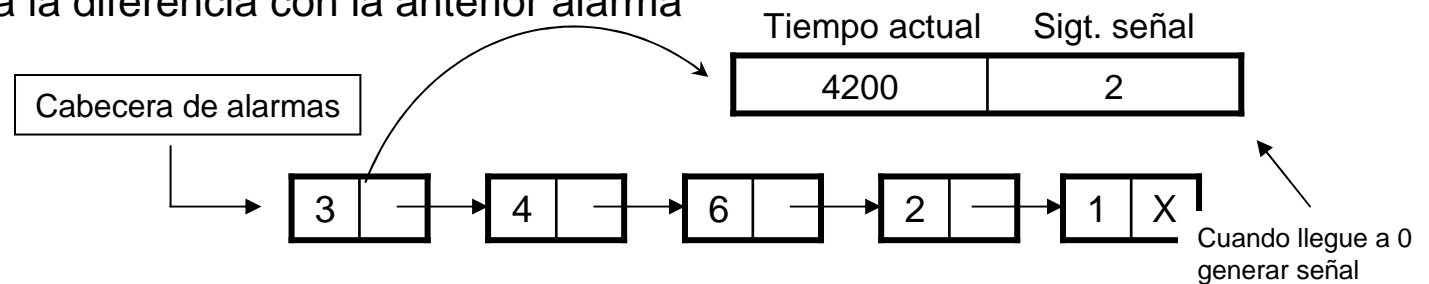
■ Alarmas

- Los procesos piden al S.O. que los informe en un instante det.
- Soluciones:
 - Reloj físico (si hay) por cada petición
 - Se simulan varios relojes con una tabla y un contador de alarma siguiente

		Hora actual	Sigt. señal
		183000	187000

Alarma 1	Alarma 2	Alarma N
215000	187000	207500

- Se mantiene una lista enlazada ordenada por tiempos donde cada nodo guarda la diferencia con la anterior alarma



3.2 Software para relojes (v)

- Ejercicios

- Ejercicios 1,2,3 Examen 2º parcial 05/06

- <http://ditec.um.es/so/web-0506/SolucionesTestAEjercicios2Parcial2005-5.pdf>

- Ejercicio 5 Examen junio 05/06

- <http://ditec.um.es/so/web-0506/SolucionesEjerciciosJunio2006.pdf>

- Ejercicio 4 Examen septiembre 05/06

- <http://ditec.um.es/so/web-0506/SolucionesEjerciciosSept2006.pdf>

3.2 Software para relojes (vi)

- Cronómetros guardianes (*watchdogs*)
 - Los coloca el propio S.O.
 - Por ejemplo: uso de la disquetera
 - El mecanismo utilizado es el mismo que para las alarmas solicitadas por los usuarios
 - Al agotarse el tiempo, se llama al procedimiento indicado
- Monitorización
 - El S.O. puede monitorizar distintos factores:
 - uso de memoria
 - uso de CPU
 - tiempo de E/S, etc.
 - ¡Muchas estadísticas se basan en el tiempo!

3.2 Software para relojes (vii)

Watchdog en OSO

forma
parte
del
kernel

```
unsigned int leersector (unsigned char far *buffer,  
                        char pista,  
                        char cabeza,  
                        char sector,  
                        char unidad) {  
  
    asm {cli}  
    paramotor = 36; /* Unos dos segundos */  
    asm {sti}  
    •  
    •  
    •  
}
```

```
void sai_timer(void)  
{  
    salvar_contexto;  
  
    /* Comprobar si se deben parar los motores de las disqueteras */  
    tick_paramotor();  
    •  
    •  
    •  
    restaurar_contexto;  
}
```

WatchDog de OSO (sigue)

forma
parte
del
kernel

```
void tick_paramotor(void) {
    unsigned char unidad;

    if (paramotor > 0) {
        paramotor--;
        if (paramotor == 0)
            /* Esto para los motores de las unidades de
             * disquetes */
            asm {
                mov al, 0Ch
                mov dx, 3F2h
                out dx, al
            }
    }
}
```

Índice

4. Terminales

4.1 Introducción

4.2 Terminales orientadas a carácter

4.3 Terminales con mapa en memoria

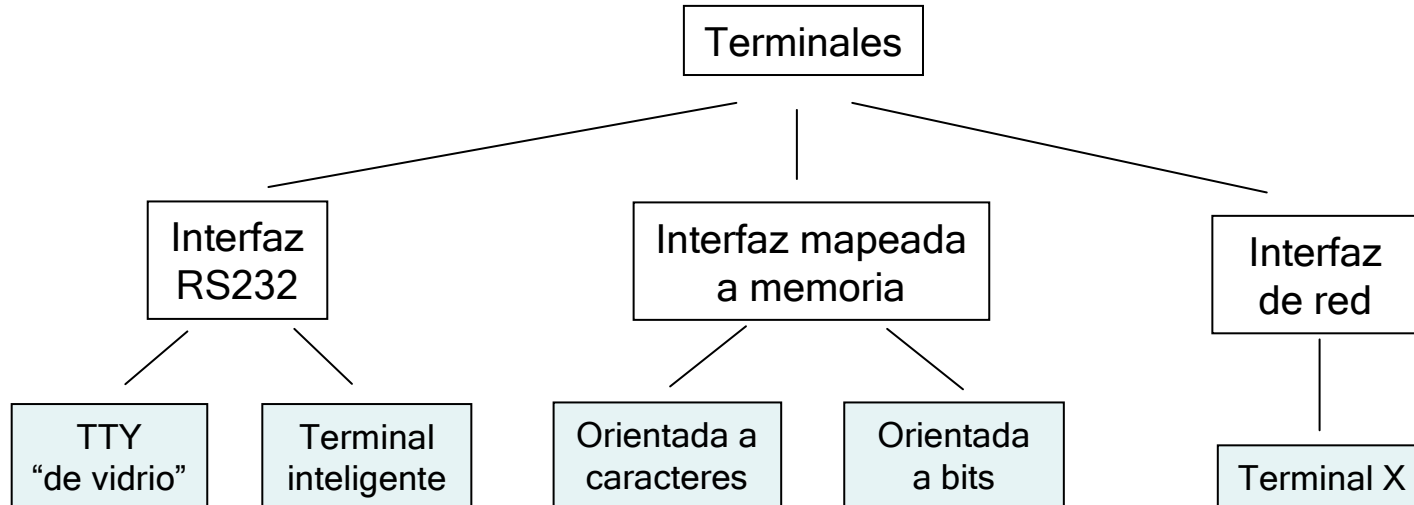
4.4 Software para la entrada

4.5 Software para la salida

Terminales

Introducción

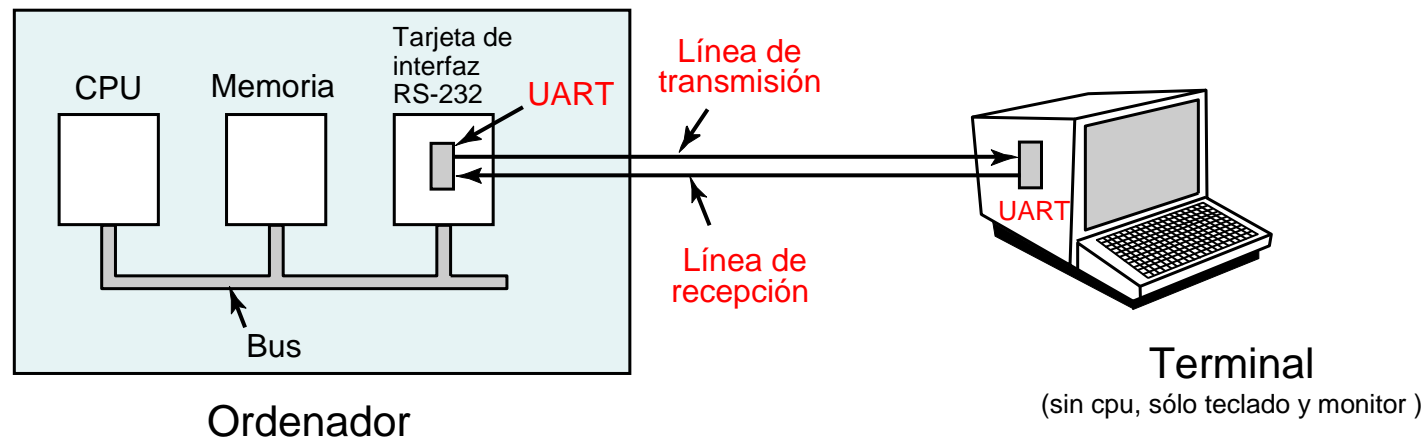
- Medio de comunicación con el ordenador
- El manejador debe ocultar las diferencias entre terminales
- Tipos
 - Terminales RS-232 (serie)
 - Terminales con mapa en memoria
 - Terminales X-Window



4.2 Terminales orientadas a carácter

Hardware de terminal RS-232

- Teclado + monitor conectados por líneas serie RS-232 a 1200, 2400, . . . 115200 baudios
- Por ejemplo, para comunicar a través de una línea telefónica
- Se siguen usando en el mundo de los *mainframes*, p.e., en sectores como las aerolíneas o los bancos
- Estas terminales operan a nivel de carácter, pero la transmisión se realiza por una línea serie: **bit a bit**
- Utilizan una UART (Universal Asynchronous Receiver Transmitter) para convertir ASCII ↔ serie

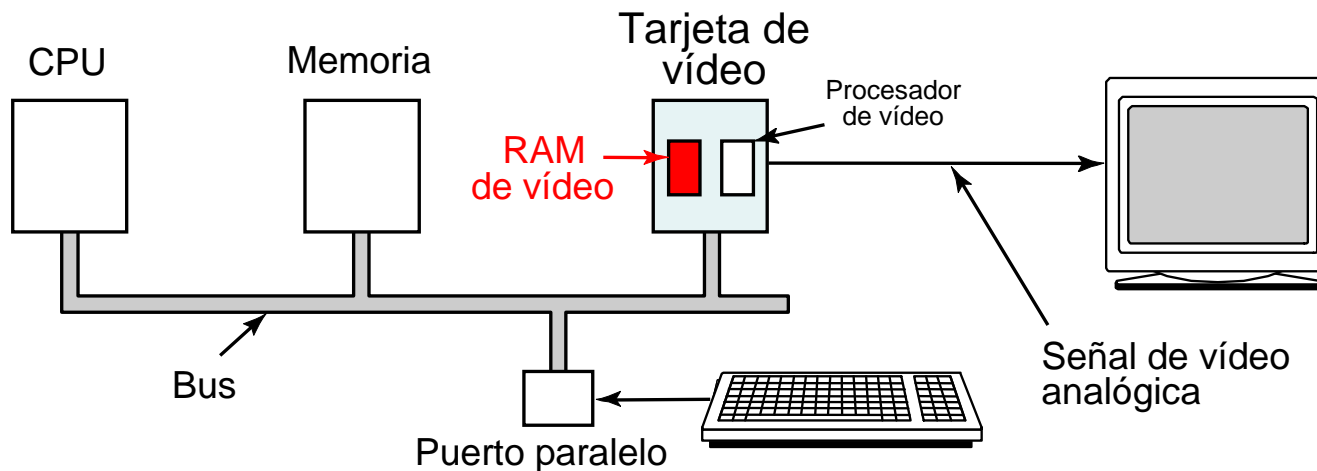


4.2 Terminales orientadas a carácter (ii)

- Para imprimir un carácter en la terminal:
 - El manejador lo escribe en la tarjeta de la interfaz, donde se guarda en un buffer
 - La UART lo transmite por la línea serie, bit a bit
 - Debido a la lenta transmisión de velocidad, el manejador se bloqueará en espera de una interrupción generada por la interfaz al transmitir el carácter
- Tipos
 - Terminales CRT no inteligentes (obsoletas)
 - Tienen un teclado para enviar caracteres al ordenador y una pantalla para mostrar los caracteres que recibe
 - Terminales CRT inteligentes
 - Son pequeños ordenadores especializados
 - Tienen CPU, memoria y programas complejos en EPROM o en
 - ROM
 - Son capaces de entender ciertas secuencias de escape (cadenas de caracteres que comienzan con un carácter ESC de ASCII y que permiten borrar la pantalla, desplazar el cursor por la pantalla, etc.)

4.3 Terminales con mapa en memoria

- Teclado + ratón + monitor
- Una zona de memoria llamada **RAM de vídeo**



- **Teclado:** la comunicación se hace por un puerto serie, paralelo o usb
 - Al pulsar una tecla, la CPU se interrumpe y el manejador de teclado extrae el carácter leyendo en un puerto de E/S
 - Sólo proporciona el número de tecla, no el código ASCII
 - El manejador debe determinar si era minúscula, mayúscula, o si estaban pulsadas las teclas ALT, CTRL, etc.

4.3 Terminales con mapa en memoria (ii)

■ Ratón

- Cada vez que se desplaza el ratón o se presiona o suelta un botón, se envía un mensaje al ordenador
- La distancia mínima es aproximadamente 0.1mm (*mickey*)
- El mensaje enviado al ordenador consta de 3 elementos:
 - Δx y $\Delta y \Rightarrow$ cambio en la posición x e y desde el último mensaje
 - Botones \Rightarrow situación de los botones

■ Pantalla

- Tarjeta de vídeo:
 - **RAM de vídeo:** forma parte del espacio de direcciones del ordenador, la CPU la direcciona igual que el resto de la memoria. La imagen de pantalla se almacena aquí
 - **Controlador de vídeo:** extrae bytes de la RAM de vídeo y genera la señal de vídeo para manejar la pantalla

4.3 Terminales con mapa en memoria (iii)

■ **Pantalla** (continúa...)

- Se divide en líneas (entre 480 y 1024) y estas a su vez en puntos entre (640 y 1200) llamados **pixels**
- Pueden estar orientadas a carácter o a bits:
 - **Terminales orientadas a caracteres** (IBM-PC): cada byte representa el código ASCII del carácter a exhibir (o 2 bytes para tener en cuenta los atributos)
 - **Terminales orientadas a bits**: cada pixel de la pantalla se representa de manera individual, mediante 1 bit para pantallas monocromo o más bits (p.e. 24 bits) para pantallas color
 - Requieren mayor cantidad de memoria aunque permiten una completa flexibilidad en los tipos y tamaños de los caracteres

4.3 Terminales con mapa en memoria (iv)

Ejemplo terminal orientada a carácter. Consola ms-dos

pinta2

```
a
mov bx,b800
mov es,bx
es:
mov byte ptr [0000],32
int 20

g

q
```

Si tecleamos debug < pinta2
en una ventana de ms-dos veremos
aparecer un 2 en la esquina superior
izquierda de la pantalla

La memoria de video en modo carácter comienza en la dirección
de memoria b800:0000.

0x32 es el código ascii de “2”

La instrucción int 20 es la llamada al sistema “exit” de ms-dos

4.4 Software para la entrada

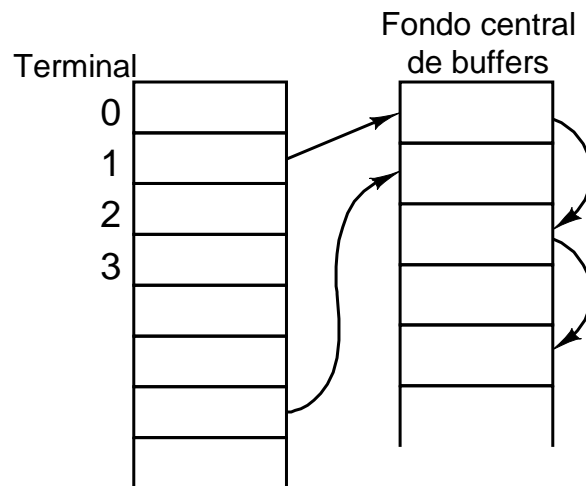
El manejador de teclado

- Pasa los caracteres del teclado a los programas de usuario
- Modos de funcionamiento:
 - Modo no interpretado (crudo):
 - El manejador se limita a aceptar entradas y pasarlas a un nivel superior sin modificación
 - El programa recibe todas las teclas: *hpla←←←ola*
 - Modo interpretado (cocinado o canónico).
 - Está orientado a líneas
 - El manejador trata ciertos caracteres especiales, se encarga de la edición de cada línea y la entrega corregida a los programas de usuarios
 - El programa recibe la línea corregida: *hola*

4.4 Software para la entrada (ii)

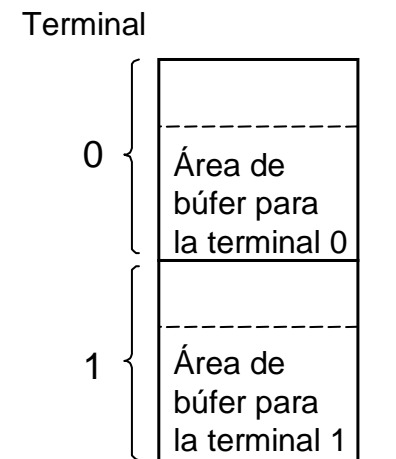
El manejador de teclado

- En cualquier modo es necesario tener *buffers* de caracteres:
 - **Fondo central de *buffers***: cada terminal está asociada a una estructura de datos que, entre otras cosas, contiene un puntero a uno de estos *buffers*
 - **Un *buffer* para cada terminal**: que se maneja de forma directa con las estructuras de datos de la misma



Estructura de datos de la terminal

(a)



Estructura de datos de la terminal

(b)

4.4 Software para la entrada (iii)

- Eco: proceso por el cual los caracteres tecleados aparecen por pantalla
 - Es complicado: un programa puede estar escribiendo en la pantalla al mismo tiempo que un usuario teclea \Rightarrow el manejador debe decidir dónde colocar las nuevas entradas para que no se sobrescriban las salidas
 - ¿Qué sucede si hay que mostrar más de 80 caracteres en una pantalla con líneas de 80 caracteres? ¿Ajustar el texto o truncar la línea?
 - Manejo de los tabuladores: normalmente se traduce por una secuencia de espacios
- Disparidad de dispositivos (¿CR ó CR/LF?): cada SO tiene su formato interno para representar los finales de línea
 - El manejador del teclado debe encargarse de convertir todas las distintas combinaciones de CR-LF al estándar interno del SO y hacer que el eco se realice de forma correcta

4.4 Software para la entrada (iv)

- Tiempo de proceso de CR y LF
 - En ciertas terminales se tarda un poco más en mostrar un CR o LF que una letra normal o un n^o, (puede ser necesario copiar un bloque de texto para desplazar el contenido de la pantalla hacia arriba, insertar «caracteres de relleno», etc.)
- Caracteres especiales: en el modo interpretado algunos caracteres tienen un significado especial

@	Elimina la línea activa
CTRL-H	Retroceder un carácter
DEL	Interrumpir el proceso (SIGINT)
CTRL-\	Provocar el vaciado de memoria (SIGQUIT)
CTRL-D	Fin de fichero

- Algunas veces esos caracteres deben utilizarse como datos ⇒ se proporciona el carácter de escape («\» en UNIX) para indicar que el siguiente carácter no se interprete

4.4 Software para la entrada (v).

Teclado en OSO. Ejemplo de bloqueo de un proceso

Las 3 entradas de OSO:

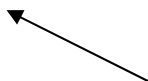
<code>void sai_llamadas(void);</code>	Instrucción int 22h
<code>void sai_teclado(void);</code>	Interrupción teclado
<code>void sai_timer(void);</code>	Interrupción de reloj

```
void sai_teclado(void)
{
    salvar_contexto;
    recoger_caracter(); /* Extrae el código de tecla del hardware de teclado, lo convierte en
                        * carácter y lo almacena en la variable "ultimocaracter" */
    asm {
        /* EOI. Decimos al PIC que la interrupción ha sido tratada. */
        mov al, 20h
        out 20h, al
    }
    restaurar_contexto;
}
```

Mueve carácter del hardware al búfer de teclado

forma parte del kernel

`static char ultimo_caracter = NULO;`



Búfer de teclado de un solo byte

Servicio “retirar carácter del búfer de teclado (se deja en el búfer de usuario)”

Aplicación

```
int read(int fd, char far * buffer, int count)
{
    asm {
        mov ah, 1
        mov cx, count
        mov dx, fd
        les bx, dword ptr buffer
        int 22h
    }
}
```

el compilador
retorna el
entero en ax

kernel

```
void sai_llamadas(void)
{
    salvar_contexto;
    current->contexto->ax =
        (tabla_llamadas[current->contexto->ax >> 8].func)();
    restaurar_contexto;
}
```

```
struct syscall
{
    int (*func) (void);
};

struct syscall tabla_llamadas[] =
{sys_crear_proceso,
  sys_leer,
  sys_escribir};

/* Crear un proceso */
/* Leer de un fichero */
/* Escribir en un fichero */
```

salta a sys_leer

Servicio “retirar carácter del búfer de teclado (sigue)

```
static int sys_leer(void)
{
```

```
    char far * buffer;
    char c;
```

```
    while (!(c = obtener_caracter()))
```

```
    {
        current->estado = BLOQ_TECLADO;
        ceder_CPU();
    }
```

Si el búfer de teclado está vacío el proceso se bloquea (cede CPU avisando previamente que no se le ceda a él la CPU). Cuando llegue la Interrupción de teclado (cuando se teclee el carácter) ésta introducirá el carácter en el búfer de teclado y cambiará el estado del proceso a listo (lo realiza la función recoger_caracter)

```
    buffer = (char far *)(((long)current->contexto->es << 16) +
                        current->contexto->bx);
```

Apunta variable buffer al búfer del usuario

```
    *buffer = c;
```

Deja carácter en el búfer del usuario

```
    /* Devolvemos el total de caracteres leídos */
    return 1;
```

```
}
```

```
/* Devuelve el carácter almacenado en la variable “ultimo_caracter” */
```

```
char obtener_caracter(void) {
```

```
    char c = ultimo_caracter;
```

```
    ultimo_caracter = NULO;
```

```
    return c;
```

```
}
```

forma
parte
del
kernel

¿Deberán estar las interrupciones permitidas en este momento?

4.5 Software para la salida

- Salida en terminales RS-232
 - Hay *buffers* asociados a cada terminal
 - La salida se copia primero en los *buffers*
 - Tras copiar la salida, se envía el primer carácter y el manejador se duerme. Cuando llega una interrupción, se envía el siguiente carácter, .
..
- Salida en terminales con mapa en memoria
 - Los caracteres se colocan en la memoria de vídeo
 - Hay que tratar ciertos de forma especial (CR, LF, Ctrl-G, etc.)
 - Se debe controlar la posición actual en la memoria de vídeo ⇒ el siguiente carácter se coloca en esa posición y se avanza a la siguiente

4.5 Software para la salida (ii)

- Salida en terminales con mapa en memoria (continúa. . .)
 - Al procesar LF en la parte inferior de la pantalla, hay que desplazar el contenido hacia arriba:
 - La controladora posee un registro que determina la posición dentro de la memoria de vídeo donde empieza la línea superior
 - Desplazar el contenido de la pantalla \Rightarrow actualizar el registro
 - Controlar el cursor:
 - La controladora posee un registro que determina la posición dentro de la memoria de vídeo
- Servicios del manejador
 - A partir de secuencias de escape: cursor arriba, desplazar pantalla, insertar un carácter o una línea en la posición del cursor, desplazar el contenido de la pantalla “n” líneas, etc.

Índice

5. E/S en Linux

5.1 Conceptos fundamentales

5.2 Implementación de E/S en Linux

5.1 Conceptos fundamentales

- Casi todos los dispositivos de E/S se representan como **ficheros especiales**
 - /dev/hda1 para la primera partición del primer disco IDE
 - /dev/lp0 para la impresora
- El acceso a estos ficheros especiales es mediante las llamadas al sistema *read* y *write*
- Para cada fichero especial hay asociado un manejador de dispositivo
- Cada manejador tiene un **número de dispositivo principal** (*major*) que sirve para identificarlo
- Si el manejador sirve a varios dispositivos, cada dispositivo tiene un **número de dispositivo secundario** (*minor*) que lo identifica
- Juntos, el n^o principal y el secundario especifican de forma única cada dispositivo de E/S

5.1 Conceptos fundamentales (ii)

- Dos tipos de ficheros especiales:
 - **Fichero especial de bloques** ⇒ dispositivos de bloques
 - **Fichero especial de caracteres** ⇒ dispositivos de caracteres
- Los dispositivos de bloques
 - Incluyen discos y cintas
 - Direccionamiento directo utilizando bloques
 - El manejador aísla al resto del sistema de pistas, cilindros, etc.
 - Acceso directo (como en /dev/fd0) o a través del S.F.
 - Memoria caché de *buffers*
- Los dispositivos de caracteres
 - Terminales, impresoras y otros que no usan la caché de *buffers*
 - Utilizan también una pequeña memoria intermedia ⇒ **listas-C**

5.2 Implementación de E/S en Linux

- La E/S en Linux se implementa como una colección de manejadores, uno por tipo de dispositivo, que aísla al resto del SO de las peculiaridades del HW
- Cada manejador se divide en dos partes:
 - La mitad superior se ejecuta en el contexto del invocador y se comunica con el resto del SO
 - La mitad inferior se ejecuta en el contexto del *kernel* e interactúa con el dispositivo
- Los manejadores pueden invocar procedimientos del *kernel* para asignar memoria, administrar temporizadores, controlar DMA, etc.

5.2 Implementación de E/S en Linux (ii)

- Manejadores para dispositivos de bloques
 - Objetivo: reducir al mínimo el nº de transferencias reales efectuadas
 - Para ello, existe en memoria principal una *caché de buffers* (disco o bloques) entre el sistema de ficheros y los manejadores de disco (visto en el tema 5)
 - Cuando se necesita un bloque de disco por cualquier motivo (nodo-i, directorio o datos), primero se verifica si está en el *caché de buffers*. Si está se toma, evitando un acceso a disco
 - Si el bloque no está, se lee del disco, se coloca en la caché y de allí se copia a donde se necesita
 - Se utiliza tanto en las lecturas como en las escrituras
 - Para las escrituras, un demonio se encarga de realizar la actualización en disco de los bloques modificados de forma periódica

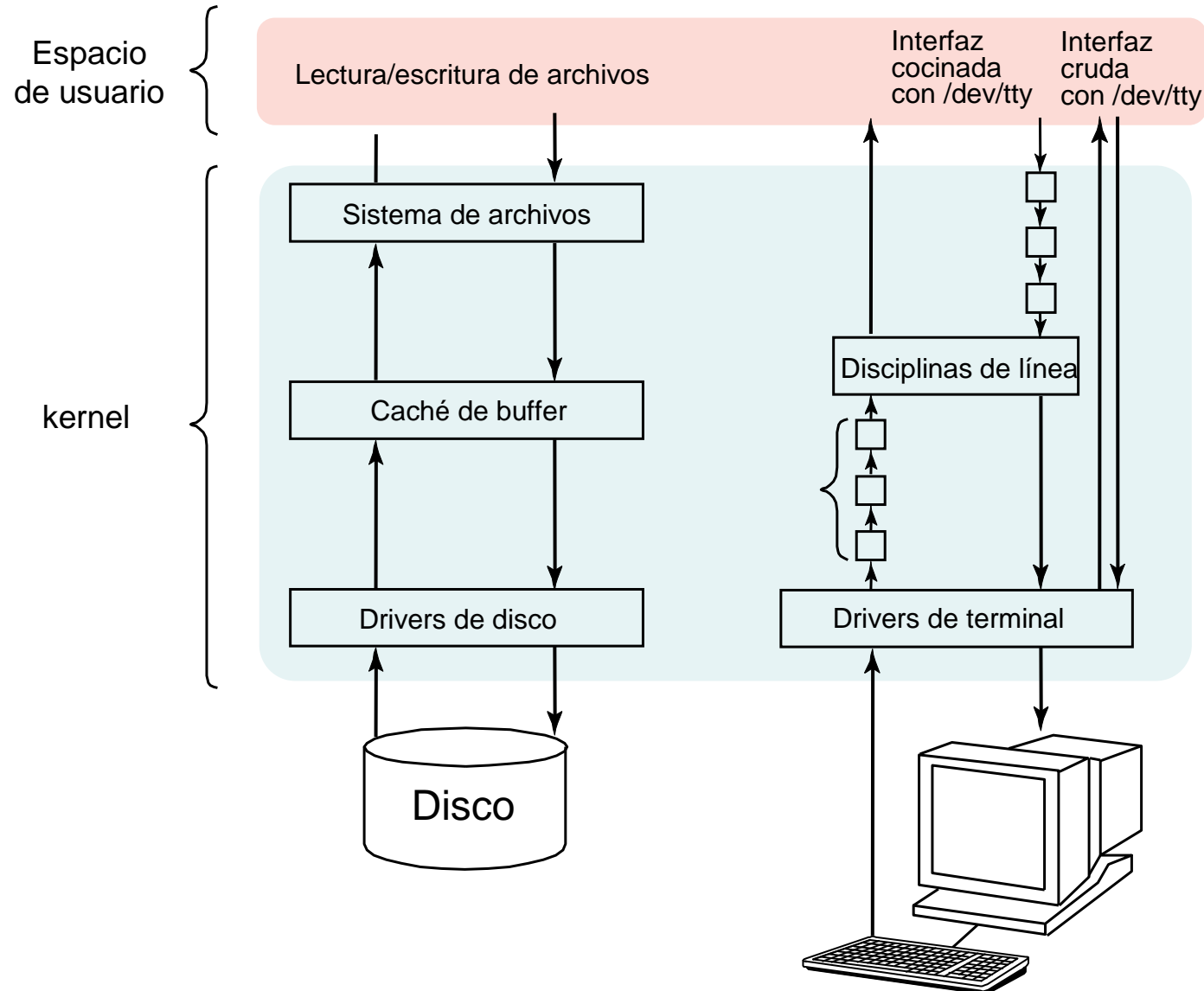
5.2 Implementación de E/S en Linux (iii)

- Manejadores para dispositivos de caracteres
 - Sistema de *buffers* para caracteres
 - Se utilizan unas estructuras de datos llamada **listas C**, formadas por un bloque de hasta 64 caracteres, un contador y un puntero al siguiente bloque
 - Se tienen dos colas: cola directa y cola canónica
 - El paso de la cola directa (flujo **no interpretado** de caracteres) a la cola canónica (flujo **interpretado** de caracteres) se activa con el retorno de carro
 - Si el proceso quiere, puede recoger un carácter cada vez que llega uno nuevo \Rightarrow acceso directo, no interpretado o crudo

5.2 Implementación de E/S en Linux (iv)

- Manejadores para dispositivos de caracteres (continúa...)
 - Entrada de caracteres:
 - Al llegar caracteres se colocan en la cola directa
 - Los caracteres se pasan por un fragmento de código del *kernel* llamado **disciplina de líneas**, que actúa como filtro, aceptando caracteres en modo no interpretado, procesándolos y produciendo lo que se conoce como *flujo de cocinado de caracteres*
 - Salida:
 - Funciona de manera similar: expandiendo tabulaciones a espacios, añadiendo caracteres de relleno, etc.
 - Las salidas pueden pasar por la disciplina de líneas (modo interpretado) o evitarla (no interpretado)
 - La salida en modo no interpretado es útil para cuando se quiere enviar datos binarios a otros ordenadores

5.2 Implementación de E/S en Linux (v)



Índice

- 6. E/S en Windows 2000 (Tanenbaum [C11.6.1, C11.6.3 y C11.6.4] y «Win32 System Programming» de Johnson Hart [C14])
 - 6.1 Conceptos fundamentales
 - 6.2 Implementación de E/S en Windows 2000

6.1 Conceptos fundamentales

- El objetivo del sistema de E/S de Windows 2000 es proporcionar un marco dentro del cual se maneje con eficiencia una amplia variedad de dispositivos de E/S y sea fácil agregar nuevos dispositivos
- **El administrador de E/S** es el responsable de todas las operaciones de E/S para el sistema operativo
 - Es responsable de los sistemas de ficheros, del administrador de caché, de los manejadores de dispositivos y de los manejadores de los protocolos de red
 - Controla qué sistemas de ficheros están instalados y cargados
 - Maneja los *buffers* para las peticiones de E/S
 - Trabaja con el manejador de memoria virtual para proporcionar E/S de ficheros mapeados en memoria

6.1 Conceptos fundamentales (ii)

- El administrador de E/S colabora de forma estrecha con el **administrador de Plug and Play** que detecta qué dispositivos están conectados, asigna recursos hardware (como niveles de interrupción), localiza los manejadores apropiados y los carga en memoria
- El administrador de E/S también está relacionado con el **administrador de consumo eléctrico**, encargado del encendido y apagado de los dispositivos que permiten gestionar el bajo consumo
- Tb está el **administrador de caché**. Este proporciona una caché centralizada que puede ser usada por todos los componentes que están bajo el control del administrador de E/S. En especial, la caché la usan los sistemas de ficheros y los componentes de red

6.1 Conceptos fundamentales (iii)

- Windows 2000 realiza un manejo de la E/S síncrona o asíncrona.
- En la E/S asíncrona un subproceso puede iniciar una operación y luego seguir ejecutándose en paralelo con la E/S. Los subprocesos cuentan con varias formas de averiguar si ya finalizó la E/S:
 - Esperar en un *handle*. El núcleo activa un indicador asociado a un *handle* cuando una operación sobre ese *handle* termina. El subproceso puede esperar en el *handle* para averiguar cuándo termina la operación de E/S
 - Esperar en un objeto evento. Permite múltiples solicitudes simultáneas sobre un mismo dispositivo o fichero. El hilo crea un evento por cada solicitud de E/S que realice. A posteriori puede esperar en uno o varios eventos para averiguar cuándo terminan las solicitudes
 - E/S extendida o alertable. Hace uso de una cola conocida como *Llamada a Procedimiento Asíncrono*. El hilo realiza una solicitud de E/S indicando que cuando finalice se invoque a una rutina que permitirá el tratamiento de los datos recibidos. Dicha rutina se almacena en la cola anterior
 - Puertos de finalización de E/S. Un puerto es un objeto al que se asocian un conjunto de *handles* e hilos. Al terminar una operación de E/S sobre uno de los *handles* se despierta a uno de los hilos para atenderlo

6.2 Implementación de E/S en Windows 2000

- La estructura básica del sistema de E/S en Windows está formada por un conjunto de procedimientos independientes del dispositivo, que se encargan de ciertos aspectos de la E/S, y un conjunto de manejadores de dispositivos, que se cargan en memoria para comunicarse con los dispositivos
- Para un funcionamiento correcto, los manejadores de dispositivo deben ajustarse al **modelo de manejadores de Windows** (Windows Driver Model, WDM) que define Windows 2000
- Windows 2000 proporciona un conjunto de herramientas, llamado *Driver Development Kit* (DDK), que ayuda a producir manejadores de dispositivo que se ajustan al modelo anterior

6.2 Implementación de E/S en Windows 2000 (ii)

- Los manejadores que se ajusten al **modelo de manejadores de Windows** deberán cumplir los siguientes requisitos:
 1. Manejar solicitudes de E/S con un **formato estándar**. Las solicitudes de E/S tienen el formato de un paquete estandarizado llamado **paquete de solicitud de E/S** (IRP; *I/O Request Packet*). Los manejadores deberán aceptarlos y procesarlos
 2. Estar basados en **objetos**, en el sentido de reconocer una lista específica de métodos que puede invocar el resto del sistema, así como poder interactuar con otros objetos
 3. Permitir que se **añadan** o **quiten** dispositivos *Plug-and-Play* dinámicamente. Si el dispositivo se añade o quita de repente al sistema, el manejador debe estar preparado para aceptar esta información y actuar en concordancia

6.2 Implementación de E/S en Windows 2000 (iii)

- **Modelo de manejadores de Windows** (continúa. . .)
 4. Permitir la administración de **consumo eléctrico**, en su caso. Si el sistema cambia a modo de hibernación con bajo consumo, los dispositivos capaces de efectuar el cambio deberán hacerlo para ahorrar energía, así como despertar cuando se les indique
 5. Ser **configurables** desde el punto de vista del uso de los recursos, e.d., no incluir valores fijos de IRQ's o puertos de E/S
 6. Ser **reentrantes** para usarse en multiprocesadores. El manejador debe funcionar de forma correcta aunque esté siendo ejecutado al mismo tiempo por 2 o más CPU's. Por tanto, el acceso a las estructuras de datos delicadas debe hacerse de forma restringida
 7. Ser **portables** entre Windows 98 y Windows 2000 (aunque haya que recompilar el manejador en cada caso)

6.2 Implementación de E/S en Windows 2000 (iii)

- El administrador de E/S convierte las peticiones que recibe en un paquete IRP y envía dicho paquete al manejador de dispositivo correcto para que lo procese
- Cuando la operación ha terminado, el administrador de E/S recibe el IRP del manejador del dispositivo y completa la operación
- ¿Cómo se localizan los manejadores a usar?
 - Windows detecta de forma automática los dispositivos e invoca al administrador de *Plug and Play*
 - El administrador de *Plug and Play* sondea el dispositivo para averiguar el fabricante y el número de modelo. Con estos datos comprueba si tiene un manejador para ese dispositivo
 - Si encuentra el manejador lo carga en memoria
 - Si no lo encuentra, pide al usuario que le indique dónde (disquete o CD-ROM) puede encontrarlo

6.2 Implementación de E/S en Windows 2000 (iv)

- Entre los procedimientos que debe soportar un manejador de dispositivo están:
 - *DriverEntry*: asigna valores iniciales al manejador y se invoca inmediatamente después de cargarlo. Podría realizar la inicialización de estructuras de datos del manejador, pero no se comunica con el dispositivo
 - *AddDevice*: se invoca una vez por cada dispositivo que va a añadirse (lo llama el administrador de *Plug and Play*)
 - A continuación se invoca al manejador con el primer paquete IRP que establece el vector de interrupción y asigna valores iniciales al hardware
 - Otros procedimientos son el de servicio de interrupción, procedimientos para administrar temporizadores, control de DMA, etc.

6.2 Implementación de E/S en Windows 2000 (v)

- Un manejador puede realizar todo el trabajo él solo, pero también es posible apilar manejadores.
- En caso de tener manejadores apilados, la solicitud podría pasar por una serie de manejadores, cada uno de los cuales realiza una parte del trabajo:
 - Un uso común de manejadores apilados, sería para separar el manejo del bus del dispositivo en sí
 - Otra posibilidad es tener la capacidad de insertar **manejadores filtro**, que aplicarían alguna transformación a los datos como, por ejemplo, comprimir o cifrar los datos