Sistemas Operativos

Unidad 9: Gestión de Entrada Salida.

"Los discos duros me han decepcionado más que la mayoría de las tecnologías."

~ Steve Wozniak

Gestión de Entrada Salida

Introducción

Probablemente la E/S es el aspecto más complicado en el diseño de un sistema operativo y esto se debe a la gran variedad de dispositivos (Discos, teclados, mouse, pantalla, etc) que hace difícil establecer una solución general y uniforme.

Evolución del sistema de E/S

Conforme se fueron dando los avances tanto en los componentes como en los sistemas operativos, la complejidad y sofisticación de estos componentes individuales se fue incrementando. Estos avances y mejoras se pueden resumir en las siguientes etapas:

Técnica	Etapa	Descripción
E/S Programada	1era	El procesador controla directamente un dispositivo de E/S y es el encargado de mover los datos desde y hacia la memoria principal.
	2da	Se implementan interfaces en los dispositivos de E/S, lo que permite que el procesador se pueda independizar de los detalles de cada dispositivo. El procesador espera que se complete la operación de E/S y luego mueve los datos desde/hacia la memoria principal.
Uso de interrupciones	3era	Con el uso de interrupciones se logra que el procesador sólo participe al comienzo y al final de la operación de E/S, posibilitando la multiprogramación ya que el procesador puede ejecutar otro proceso mientras se realiza la transferencia.
DMA	4ta	A los módulos de E/S se les otorga el control directo de la memoria principal para que puedan mover un bloque de datos desde/hacia la memoria principal sin intervención del procesador. El procesador es interrumpido sólo al principio y al final de la operación.
	5ta y 6ta	Se mejora el módulo de DMA hasta convertirlo en un procesador independiente con su propio conjunto de instrucciones. El módulo de DMA puede ahora ejecutar procesos del SO para realizar la E/S.

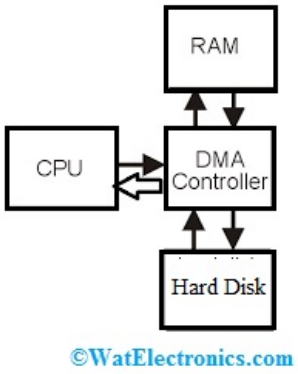
Nota: tanto la 1er como la 2da etapa “adolecen” de los que se llama espera activa ya que el proceso en ejecución debe esperar a que se transfieran los datos.

DMA

DMA o acceso directo a memoria (por sus siglas en inglés) es una tecnología que mejora la eficiencia de la transferencia de datos, que resulta muy conveniente cuando se requiere transferir grandes volúmenes de datos o simplemente para mejorar la eficiencia del CPU ya que se libera al mismo de la tarea de transferencia.

Cabe destacar que para implementar esta tecnología, se requiere de un módulo de hardware específico llamado DMAC (DMA controller o controlador de DMA). El DMAC suele estar integrado en el chipset de la computadora, que incluye los chips *northbridge* y *southbridge*. Dependiendo del diseño del sistema, el DMAC puede formar parte del chip northbridge, del chip southbridge o ser un chip (independiente) dedicado. En algunos chipsets modernos, este módulo puede estar integrado directamente en la CPU.

Un esquema simplificado de esta tecnología sería el siguiente:



¿Cómo funciona el acceso directo a memoria (DMA)?

Paso 1. Inicio

El inicio de la transferencia puede ser disparado por un proceso o bien por un dispositivo. Un proceso utilizará un *system call* para notificar al sistema operativo que necesita enviar o recibir datos desde un dispositivo.

Cuando un dispositivo necesita enviar o recibir datos de la memoria, inicia una solicitud DMA.

Una vez que se acepta una solicitud, el controlador DMA toma control temporalmente del bus del sistema para facilitar la comunicación directa entre dispositivos y la memoria.

Paso 2. Solicitud

Cuando un dispositivo necesita transferir datos desde o hacia la memoria principal, envía una señal de solicitud DMA al controlador DMA. Esta señal indica que la CPU no es necesaria para esta operación particular de transferencia de datos.

Al recibir la solicitud DMA, el controlador DMA verifica si el bus está disponible y luego inicia el acceso a la memoria. Al manejar estas solicitudes independientemente de la CPU, se reduce significativamente la sobrecarga del procesador y se aceleran las transferencias de datos entre dispositivos y memoria.

El controlador DMA gestiona la temporización y la priorización de estas solicitudes mediante técnicas de arbitraje eficientes. Esto asegura que múltiples dispositivos puedan comunicarse con la memoria sin problemas, sin causar conflictos o cuellos de botella en el flujo de datos.

Paso 3: Arbitraje

Cuando varios dispositivos necesitan acceder a la memoria simultáneamente, el controlador DMA arbitra entre estas solicitudes para garantizar una utilización eficiente de los recursos del sistema.

A través del arbitraje, el controlador DMA prioriza y programa tareas de transferencia de datos en función de reglas o algoritmos predefinidos. Este proceso ayuda a prevenir conflictos y asegura que cada dispositivo tenga un acceso justo al bus de memoria sin causar cuellos de botella o retrasos en las transferencias de datos.

Paso 4: Acceso a la memoria

Una vez que el controlador DMA obtiene el control del bus del sistema, puede acceder directamente a la memoria sin involucrar a la CPU. Esta interacción permite transferencias de datos eficientes y rápidas entre periféricos y la memoria.

Los datos se leen o escriben en direcciones de memoria específicas según las instrucciones que recibió el controlador DMA. El controlador asegura que los datos se transfieran con precisión y rapidez sin requerir la intervención constante de la CPU.

Paso 6: Transferencia de datos

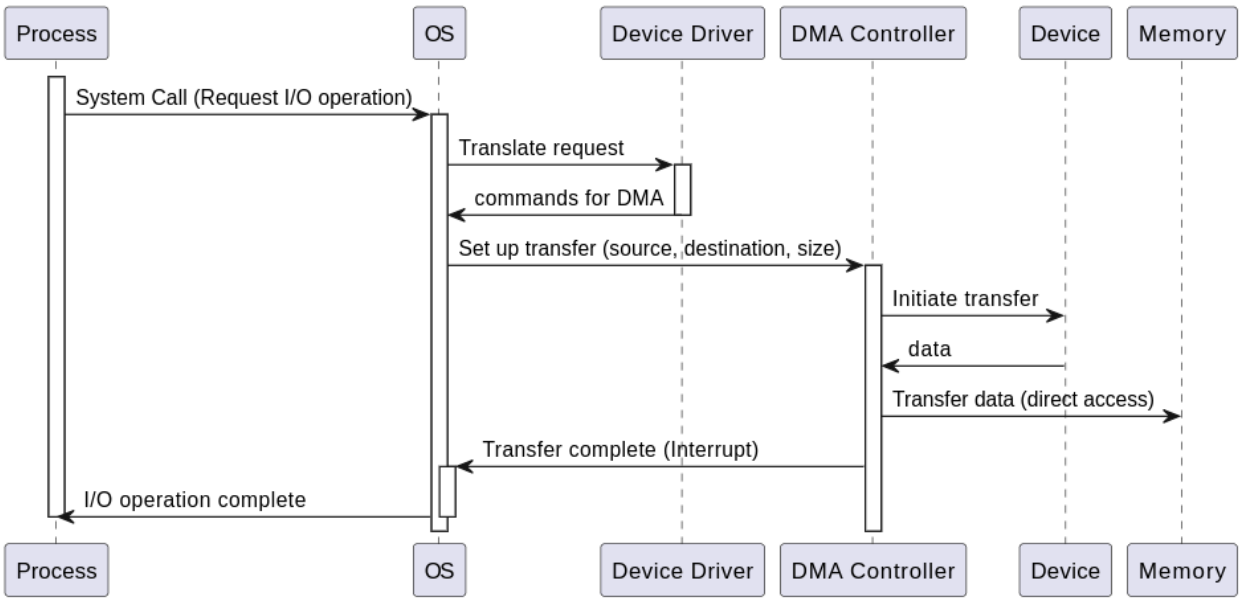
Una vez que el controlador DMA tiene control del bus, inicia el movimiento real de datos entre dispositivos y memoria. El controlador DMA se coordina con los dispositivos de origen y destino para transferir datos de manera eficiente sin involucrar a la CPU. Durante la transferencia de datos, la información fluye directamente de un dispositivo a otro a través de canales DMA sin intervención de la CPU.

Paso 7: Finalización e interrupción

Una vez completada la transferencia de datos, el controlador DMA activa una interrupción para notificar a la CPU. Esta interrupción indica que la operación DMA ha finalizado correctamente. Luego, la CPU puede reanudar sus tareas o manejar cualquier acción de seguimiento necesaria basada en la finalización de la transferencia de datos.

Al recibir una interrupción del controlador DMA, la CPU puede ejecutar rutinas o procedimientos específicos configurados para manejar tareas posteriores a DMA, como actualizar indicadores de estado, notificar a los componentes de software sobre la disponibilidad de datos o iniciar pasos de procesamiento adicionales basados en los datos transferidos.

A continuación se presenta un diagrama de secuencia simplificado de estos pasos:



Uso de buffers

Además de las complejidades que presenta manejar un dispositivo de E/S, este tipo de operaciones conlleva una serie de cuestiones de diseño del sistema operativo que permitan garantizar cierta eficiencia e imponer la menor sobrecarga posible. Imaginemos que un proceso requiere transferir datos desde el disco hacia la memoria principal. En ese punto el proceso debe esperar a que los datos estén disponibles para continuar su ejecución. Esta espera puede ser activa o utilizando interrupciones.

Esta situación presenta 2 problemas:

- el proceso debe esperar por la operación de E/S
- Si el sistema operativo decide suspender el proceso, esto interfiere con la política de reemplazo (ver capítulo de memoria virtual)

El 2do punto se debe a que el sistema operativo no puede reemplazar todas las páginas del proceso suspendido, al menos la página donde se van a almacenar los datos debe permanecer en memoria, de no ser así se pueden perder los datos o producirse un caso de interbloqueo de un único proceso.

Si el proceso lanza una operación de E/S y se suspende, el proceso se bloquea esperando el evento de E/S y la operación de E/S se bloquea esperando a que el proceso se cargue en memoria. Para evitar esta situación es que la página del proceso que se utilizará para recibir los datos debe permanecer en memoria. La misma situación se produce en caso de que la operación sea de salida y no de entrada de datos.

Para evitar estos problemas, es conveniente que se implementen las operaciones de E/S con buffers.

El uso de buffers es una técnica que si bien no permitirá que un dispositivo de E/S funcione a la velocidad del procesador o de la memoria principal, amortigua los picos en la demanda de E/S. En un entorno multiprogramado, el uso de buffers es una técnica que puede incrementar la eficiencia del sistema operativo y el rendimiento de los procesos individuales.

Buffer único

Para dispositivos orientados a bloques de datos, como los discos, las transferencias de entrada utilizan el buffer del sistema. Cuando se completa la transferencia, el procesador mueve el bloque de datos al espacio de usuario y pide otro bloque. Este proceso se lo denomina lectura anticipada ya que el bloque de datos se solicita antes de que el proceso lo requiera, con lo que el proceso puede estar procesando datos mientras se lee el siguiente bloque de datos.

Esta técnica si bien permite que el sistema operativo pueda expulsar el proceso de memoria, complica la lógica del sistema operativo ya que debe mantener un registro de los buffers asignados a los procesos y por otro lado, complica la lógica de intercambio ya que si el intercambio se realiza sobre el mismo disco no tiene sentido encolar la escritura del proceso en disco. Probablemente cuando la operación de E/S finalice ya no tenga sentido expulsar el proceso.

Las mismas consideraciones aplican a operaciones de salida de bloques.

Si el dispositivo es orientado a flujo de caracteres, como una pantalla o un teclado, el esquema de buffer único se puede utilizar en modo línea a línea o byte a byte, en vez de bloques de datos.

En el caso de E/S línea a línea, el buffer es utilizado para almacenar una única línea. El proceso de usuario se suspende durante la entrada de datos, esperando la llegada de la línea completa.

Con respecto a la salida, el proceso de usuario copia la línea de salida en el buffer y continúa el procesamiento. No se necesita suspender el proceso a menos que se tenga una segunda línea y el buffer no esté vacío.

Buffer doble

Con este otro esquema, un proceso transfiere datos a o desde un buffer mientras el sistema operativo vacía o llena el otro.

El uso de buffer doble debería “suavizar” el flujo de datos entre un dispositivo de E/S y un proceso, pero si el interés está en el rendimiento de un determinado proceso, sería deseable que el dispositivo pueda sostener el ritmo del proceso. La técnica de doble buffer podría no ser la más adecuada y habría que utilizar más buffers.

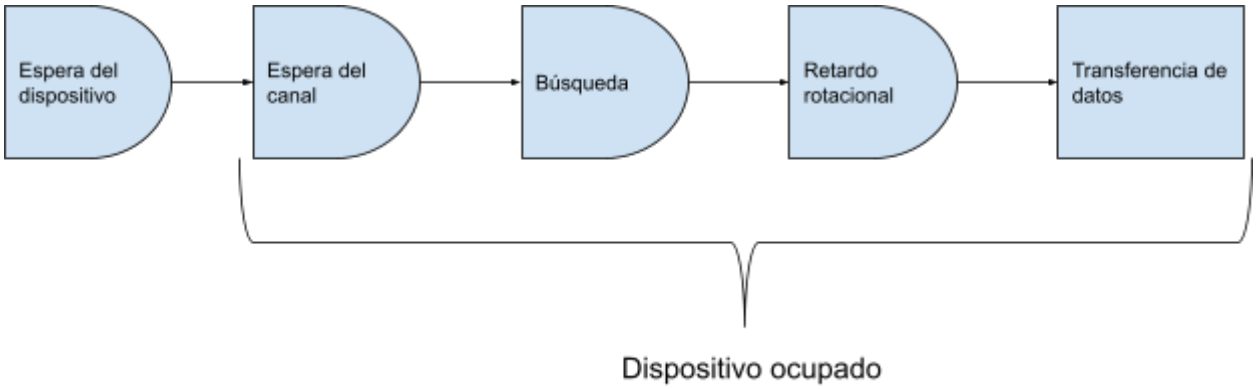
Buffer circular

Cuando se utilizan más de 2 buffers, el conjunto de buffers se los denomina buffers circular.

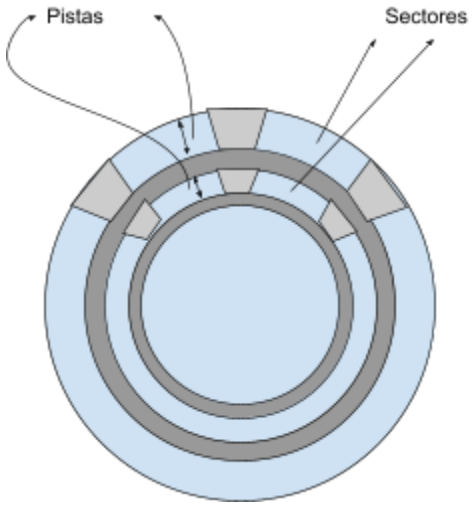
Planificación de disco

La planificación de uso del disco es de vital importancia ya que los mismos operan a una velocidad que es varias veces inferior a la velocidad de la memoria principal.

El tiempo de una transferencia depende del sistema operativo y del hardware involucrado (velocidad del bus, controlador de disco, el disco, etc). Un esquema general de los tiempos de transferencia de una operación de E/S de disco se puede representar como sigue:



Cuando el disco está en funcionamiento, se encuentra rotando a velocidad constante. Para leer o escribir, la cabeza se debe posicionar sobre la pista deseada y en el principio del sector. La selección de la pista implica el movimiento de la cabeza en un sistema de cabeza móvil o la selección de manera electrónica de una cabeza fija. En un sistema de cabeza móvil, el tiempo que se tarda en situar la cabeza en la pista se denomina **tiempo de búsqueda**. El tiempo que tarda en llegar al comienzo del sector se llama **retardo rotacional**, o latencia rotacional. La suma del tiempo de búsqueda, en caso de que lo haya, y el retardo rotacional se denomina **tiempo de acceso**, que es el tiempo que se tarda en llegar a la posición donde se debe leer o escribir. Una vez que la cabeza se posicionó, a medida que el sector se desplaza debajo de la cabeza se realiza la lectura o escritura. El tiempo que lleva realizar esta última operación se denomina **tiempo de transferencia**.



Políticas de planificación

El sistema operativo para procesar los pedidos de E/S, mantiene una cola por cada dispositivo. Los pedidos deben administrarse para que las operaciones de E/S no se realicen en orden aleatorio que sería el peor caso posible.

Los algoritmos que se describen a continuación se basan en la cantidad de movimientos que debe realizar la cabeza del disco para posicionarse en la pista solicitada.

FIFO

Es el algoritmo más sencillo para administrar las E/S. Si bien esta técnica tiene la ventaja de ser simple y equitativa, porque toda petición se acaba sirviendo, tiene un rendimiento bastante pobre en casos en los que existan muchos procesos compitiendo por el uso del disco.

Si son pocos los procesos que requieren acceso al disco y gran parte de las peticiones corresponden a sectores agrupados, entonces el rendimiento no es tan malo.

Prioridad

En un sistema basado en prioridad, la planificación del disco queda fuera del control del software de gestión de disco. Esta estrategia no se diseña para optimizar el uso del disco sino de los procesos. Se le otorga mayor prioridad a los procesos por lotes de corta duración y a los trabajos interactivos que a los trabajos largos que requieren más procesamiento. Esto permite que muchos trabajos cortos salgan del sistema rápidamente y proporciona buenos tiempos de

respuesta en trabajos interactivos. La desventaja es que los trabajos largos deben esperar, y hasta sufrir *starvation* en caso de que se agreguen al sistema procesos con mayor prioridad.

SSTF

Primero el de tiempo de servicio más corto, esta política consiste en seleccionar la petición de E/S del disco que requiera el menor movimiento del brazo desde su posición actual. De esta manera se busca que el tiempo de búsqueda sea mínimo. Este algoritmo proporciona un rendimiento mejor que FIFO. Como el brazo puede moverse en ambas direcciones, en los casos de empate se puede utilizar un algoritmo aleatorio.

SCAN

A excepción de la política FIFO, el resto de las políticas pueden dejar alguna petición sin servir. La política SCAN resuelve este problema (*starvation*). Con esta política, el brazo sólo debe moverse en 1 dirección satisfaciendo todas las peticiones que encuentre en su camino, hasta alcanzar la última pista en esa dirección o hasta que no haya más peticiones en esa dirección. En ese momento la dirección del brazo se invierte y la búsqueda continúa en la posición opuesta hasta agotar todas las peticiones.

Este algoritmo no aprovecha la proximidad como sí lo hace SSTF ya que una vez que pasó por una pista, deberá agotar todas las peticiones en esa dirección y recién cuando haga el barrido en la dirección opuesta atenderá a los pedidos que se encontraban cercanos a la pista recientemente dejada.

C-SCAN

C-SCAN o Circular SCAN restringe la búsqueda a una sola dirección , por lo tanto, luego de visitar la última pista en una dirección, el brazo vuelve al extremo opuesto del disco. Esto permite reducir el retardo máximo que pueden experimentar las nuevas peticiones con SCAN.

SCAN-N y FSCAN

Con SSFT, SCAN y C-SCAN es posible que el brazo no pueda moverse durante un período de tiempo considerable si existen procesos que tienen elevadas tasas de acceso a una determinada pista.

SCAN-N divide la cola de peticiones del disco en varias colas de longitud N. En cada momento se procesa sólo 1 cola utilizando SCAN. Mientras se procesa una cola, las nuevas peticiones se añaden a otra cola.

FSCAN aplica la misma política, sólo que maneja 2 colas únicamente.

Cache de disco

Una cache de disco es un buffer en memoria principal para almacenar sectores del disco. La cache contiene una copia de algunos de los sectores del disco. Cuando se hace una petición de E/S solicitando determinado sector, se comprueba si el sector está en la cache del disco. En caso afirmativo, se sirve la petición desde la cache. Debido al fenómeno de proximidad de referencias, cuando se lee un bloque de datos en la cache para satisfacer una única petición de E/S, es probable que haya referencias a ese mismo bloque en el futuro.

Consideraciones de diseño

El primer aspecto a considerar es cómo entregar los datos desde la cache a un proceso que los solicita. Se pueden abordar dos estrategias, la primera es copiar el bloque de datos desde la cache hasta la memoria asignada al proceso que lo solicita y la segunda y más performante es pasar un puntero al bloque de datos de la cache utilizando técnicas de memoria compartida.

El segundo aspecto de diseño está relacionado con la estrategia de reemplazo. Cuando se trae un nuevo sector a la cache de disco, se debe reemplazar uno de los bloques existentes.

El algoritmo más utilizado para este fin es LRU, se reemplaza el bloque que ha estado en la cache más tiempo sin ser accedido.

Otro algoritmo utilizado es el **menos frecuentemente usado (LFU)**, donde se reemplaza el bloque del conjunto que ha experimentado la menor cantidad de referencias. Cuando se trae un bloque a memoria, se le asigna un contador de referencias con el valor 1. Este contador se incrementa cada vez que el bloque recibe una referencia. Cuando se requiere un reemplazo, se selecciona el bloque con un contador más pequeño. Este algoritmo posee una deficiencia que es la siguiente: si las referencias a memoria se producen en intervalos de tiempo muy cortos, el contador de referencias no se incrementa, por lo que el principio de proximidad puede causar errores en el algoritmo LFU.

Una mejora a este algoritmo propone tener 3 pilas: nueva, intermedia y antigua donde se guarden las referencias a los bloques. Cuando se debe reemplazar un bloque, sólo se seleccionan bloques de la pila antigua. Esta modificación presenta mejores resultados que LRU y LFU.

RAID

Con el objetivo de seguir mejorando el rendimiento de la E/S, se crean distintas configuraciones de discos que proporcionan mejoras en cuanto a la velocidad de la lectura de datos, la

redundancia y la consistencia de los datos. Esto se logra con el desarrollo de vectores de discos independientes o RAID (redundant array of independent disk).

Existen 7 niveles o configuraciones, desde RAID 0 (cero) a RAID 6, que poseen las siguientes características:

1. Conjunto de unidades físicas de disco tratado por el sistema operativo como un único dispositivo lógico.
2. Los datos se distribuyen a lo largo de las unidades físicas de un vector
3. La capacidad de redundancia del disco se utiliza para almacenar información de paridad, que garantiza la recuperación de los datos en caso de falla.

RAID 0 y 1 carecen de la 3er característica.

RAID 0

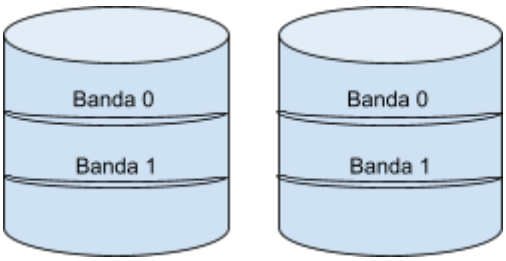
Este nivel no es un verdadero miembro de la familia RAID ya que no incluye redundancia.

En este esquema los discos se dividen en bandas (strips del inglés). Estas bandas se asignan de forma rotativa a cada disco.

La ventaja de este nivel es que los datos están distribuidos a lo largo de todos los discos del vector lo que facilita las lecturas en paralelo en el caso de que los datos solicitados no se encuentren en el mismo disco.

RAID 1

También llamado espejado (mirroring), logra redundancia duplicando los datos, a diferencia de los esquemas siguientes que lo logran aplicando cálculos de paridad. Proporciona una copia de respaldo (backup) en tiempo real.



Las principales características de este esquema son:

- ✓ Una petición de lectura puede servirse de cualquiera de los 2 discos que contienen los datos, aquel que proporcione el menor tiempo de búsqueda más la latencia rotacional.
- ✓ Proporciona tolerancia a fallos, ya que si un disco se rompe, todavía se pueden recuperar los datos del otro.

Las principales desventajas de RAID 1 son:

- ✘ El costo, ya que requiere del doble de espacio de disco. Siempre requiere una cantidad par de discos (2, 4, 6, etc).
- ✘ Una petición de escritura requiere actualizar ambas bandas. El rendimiento de la escritura lo impone el disco de mayor latencia rotacional.

RAID 2

En este esquema las bandas son muy pequeñas, pudiendo ser de un byte. RAID 2 , calcula un código de corrección de error con los bits correspondientes de cada disco de datos, almacenando los bits del código resultante en los correspondientes posiciones de bit en los múltiples discos de paridad. Normalmente se utiliza un Código de Hamming, que es capaz de corregir errores en un único bit y detectar errores en dos.

Si bien este esquema requiere menos discos que RAID 1, el costo es alto ya que requiere gran cantidad de discos redundantes. Otra desventaja es que en cada lectura se accede a todos los discos de manera simultánea y por cada lectura se verifica si hubo errores en los datos

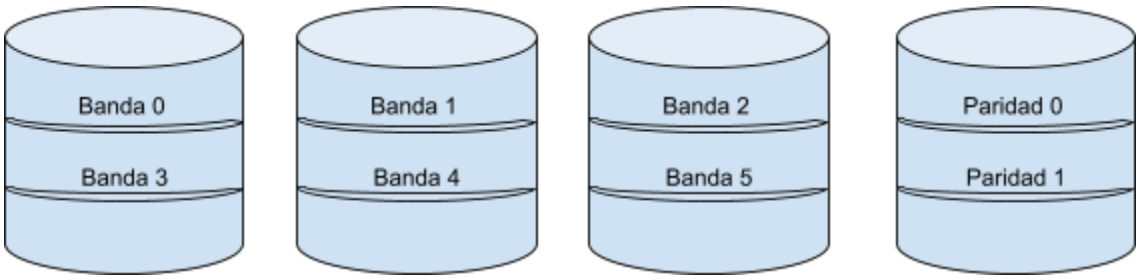
Dada la alta confiabilidad de los discos de hoy este esquema no es utilizado.

RAID 3

Similar a RAID 2, pero sólo requiere un único disco de paridad independientemente de la cantidad de discos de datos.

Este esquema guarda un bit de paridad mediante el cual puede detectar y reconstruir los datos en caso de que haya habido un error en la lectura. Esto permite reconstruir los datos de cualquier banda de datos. Este mismo esquema es aplicado en todos los esquemas siguientes.

Al igual que RAID 2 cada operación de lectura involucra todos los discos y sólo puede resolver 1 petición de lectura a la vez, por lo que no es recomendable para sistemas transaccionales.



RAID 4

Los discos de datos en este esquema son de acceso independiente, de forma que se pueden servir en paralelo peticiones de E/S independientes. Es por este motivo que RAID 4 es más adecuado para aplicaciones que requieran tasas de transferencia de datos bajas pero muchas peticiones de E/S. A diferencia de RAID 3 utiliza bandas relativamente grandes y también utiliza un único disco de paridad.

RAID 5

Es muy similar a RAID 4, la diferencia se encuentra en que las bandas de paridad se distribuyen en todos los discos. Ésta distribución evita los cuellos de botella sobre el disco de paridad.

Este esquema puede continuar funcionando a pesar del fallo de 1 de sus discos debido a que la información de paridad está distribuida en varios discos.

RAID 6

Este esquema extiende a RAID 5, generando 2 bloques de paridad en cada disco con lo cual permite el fallo de hasta 3 de sus discos y aún así seguir operando.

Este esquema utiliza 2 cálculos distintos para generar paridad.

Lectura recomendada:

- https://en.wikipedia.org/wiki/RAID

Bibliografía utilizada

● William Stallings. Sistemas operativos. Pearson Education. S.A., Madrid, 2005. ISBN-84-205-4462-0.

● AndrewS.Tanenbaum. Modern Operating System. Pearson Education Inc., 2009. ISBN-Q-lB-filBMST-L

● Multilevel Feedback queue. Wikipedia, La enciclopedia libre, 2019 [consulta: 21 de marzo del 2019]. Disponible en https://en.wikipedia.org/wiki/Multilevel_feedback_queue