**Resumen Segundo Parcial**

**Administración de Memoria Central**

**Introducción**

***def. Memoria central:*** *Está compuesta por un hardware de direccionamiento, las memorias cachés, y la memoria RAM y ROM. Adoptamos el término de memoria central para designar las cachés y las RAM y ROM como una unidad. Los espacios de almacenamiento secundarios (discos duros, lectoras de CDs/DVDs) no son parte de la memoria central.*

En un sistema monoprogramado, la administración de memoria es simple: una parte es asignada al S.O, y la otra al programa que se está ejecutando en el momento.

Pero en un sistema multiprogramado, la memoria debe dividirse en varias partes para albergar a diferentes procesos. Esta división es llevada a cabo por el S.O, y es lo que se conoce como administración o gestión de memoria.

Además, la gestión de memoria está ligada a la eficiencia del sistema en sí. Una óptima gestión de memoria nos dará como resultado un sistema operativo más eficiente.

Las características del **tiempo de acceso** y el **tiempo de ciclo de memoria** tienen una gran influencia sobre la velocidad del procesamiento.

Se define como tiempo de acceso al que transcurre entre el inicio y el fin de una operación de lectura o escritura sobre una locación de memoria. El tiempo de ciclo de memoria es el retraso que impone el hardware entre el fin de una operación y el comienzo de la siguiente.

Desde el punto de vista de la administración, la memoria central, es un vector finito de bytes que debe ser repartido entre varios procesos. La única forma de ejecutar un proceso es que el bloque de instrucciones que contiene y los datos necesarios se encuentren en memoria central.

No estamos interesados en cómo son generadas las direcciones de memoria central por un proceso, ni si son datos o instrucciones (más bien es imposible saber qué tipo de datos es observando el “ejecutable”).

Desde el punto de vista de la administración de memoria central, sólo interesa la **secuencia** de direcciones generadas por el programa durante su ejecución.

También interesa, en algunos casos, conocer la longitud del programa a ejecutar para la asignación del espacio de direccionamiento que usará el proceso en memoria central.

**Funciones del Administrador de Memoria Central**

* Mantener un registro de estado de las locaciones de memoria para saber qué zonas de memoria están ocupadas.
* Determinar una y sólo una técnica de asignación para poder hacer la selección de trabajos.
* Implementar un procedimiento de entrega y recuperación de memoria.
* Traducir las direcciones lógicas de un proceso a direcciones físicas de la memoria central con apoyo del hardware.
* Implementar protección de zonas de memoria del sistema y del usuario (es recomendable el apoyo del Hardware).

**Objetivos de la memoria central**

* **Protección:** debido a que la memoria es un recurso compartido, vamos a tener varios procesos intentando acceder a la misma. Por eso necesitamos protección en cuanto a accesos.

**Accesos a escrituras**: Ninguno de ellos puede alterar el contenido de las posiciones asignadas a otros.

**Accesos de lecturas**: privacidad.

La protección varía según el tipo de dato. Distinguimos 3 tipos:

1. **Datos puros:** se protegen contra Lectura (R) o contra escritura (W) o ambos (R/W).

2. **Código:** contra lectura (poco frecuente) y contra ejecución (X).

3. **Pila (Stack):** R (pop), W (push) y X.

Además, se distingue que el proceso:

A) No tenga permiso de acceso (se verifica la tabla de permisos y si el proceso no tiene permiso de acceso, es terminado).

B) No pueda efectuar la operación en ese instante (condiciones de concurrencia, en este caso el proceso se bloquea).

Un programa de usuario no puede acceder a ninguna porción del S.O. Tampoco pueden acceder a las instrucciones de otros procesos. Sí pueden acceder a los datos de otro proceso mediante la implementación de ciertas técnicas (memoria compartida, segmentación).

El procesador debe ser capaz de abortar cualquiera de estas instrucciones que violarían la seguridad de la memoria en el punto de ejecución.

La protección de memoria debe ser realizada por el procesador (hardware) y no por el sistema operativo (software). Esto se debe a que el sistema operativo no puede saber todas las referencias a memoria que un programa hará, y de así poder saberlo consumiría mucho tiempo en buscar las posibles violaciones de acceso a memoria.

* **Uso compartido de códigos y datos:** se requiere como condicionante para compartir una sola copia de datos o un programa que los accesos sean controlados y los códigos deben ser reentrantes o puros.

**def. Códigos reentrantes o puros:** definimos como *códigos reentrantes o puros*a aquel código de programa que no se modifica, es decir, no tiene ni usa variables globales, sino sólo locales y la almacena en una pila (stack). En otras palabras, no usa variables estáticas. Un programa informático o subrutina se llama re-entrada si puede ser interrumpido en medio de su ejecución y volver a llamarse de forma segura ("re-entrar") antes de que las invocaciones anteriores completen su ejecución. La interrupción puede ser causada por una acción interna como un salto o llamada, o por una acción externa como una interrupción o señal. Una vez que la invocación re-entrante completa, las invocaciones anteriores reanudarán su ejecución de forma correcta.

Cualquier mecanismo de protección debe tener la flexibilidad de permitir que varios procesos accedan a la misma porción de memoria. Por ejemplo, si un número de procesos están ejecutando el mismo programa, sería una gran ventaja permitir a cada proceso acceder a la misma copia del programa. Los procesos que cooperan en alguna tarea podrían necesitar un acceso compartido a las mismas estructuras de datos.

* **Reubicabilidad:** la reubicabilidad hace referencia a cómo ubicamos y re-ubicamos nuestro proceso al cargarlo en memoria central.

La ubicación es un tema importante, ya que, al momento de ejecutar nuestro programa, necesitamos saber a qué direcciones acceder (¿cuál es nuestra próxima instrucción? ¿en dónde está el valor de la variable X?).

En un sistema multiprogramado, podemos tener varios procesos en memoria central. Esto nos genera el problema de que no todos los procesos empezarán por la misma dirección inicial de memoria…

Pero también, para que nuestro sistema sea más eficiente, podemos elegir quitar procesos de memoria central y llevarlos a memoria secundaria. Cuando volvamos a cargar nuestro proceso en memoria central, su espacio de direcciones puede haber cambiado. Ahora la variable X puede encontrarse en una dirección física totalmente distinta.

Es tarea del S.O y del procesador el hecho de traducir las referencias a memoria encontradas en el programa a la ubicación actual de la memoria física.

**Generación de direcciones (resolución, address binding)**

Un programa compilado no emplea nombres simbólicos para las variables o funciones que llama; el compilador, al convertir el programa a lenguaje máquina, las sustituye por la dirección en memoria donde se encuentra la variable o la función.

Ahora bien, en los sistemas actuales, los procesos requieren coexistir con otros, para lo cual las direcciones indicadas en el código del programa pueden requerir ser traducidas al lugar relativo al sitio de inicio del proceso en memoria —esto es, las direcciones son resueltas o traducidas. Hay diferentes estrategias de resolución, que se pueden clasificar a grandes rasgos en:

* **Compile time:** las direcciones son generadas por el traductor (compilador y vinculador).

Si en el momento de la compilación se sabe donde residirá el programa en memoria puede generarse código absoluto, es decir, las direcciones de las variables y funciones serán absolutas. Si más tarde cambia la posición de inicio, será necesario volver a compilar el código. Esto es muy incómodo.

* **Load time:** al momento de compilación, las direcciones son generadas relativas al comienzo del proceso. Es decir, se utiliza un registro base que indica la dirección de inicio del programa, y un offset que indica cuánto hay que desplazarse para hallar el dato.

Pero la dirección de inicio es conocida solamente por el loader, ya que este será el encargado de decir dónde se cargará el programa. Conociendo este registro base, es tarea del loader resolver la cuenta Registro Base + Offset para traducir las direcciones lógicas en direcciones absolutas.

Las direcciones son resueltas por el loader antes de iniciar la ejecución.

* **Execution time:** el programa nunca hace referencia a una ubicación absoluta de memoria, sino que lo hace siempre relativo a una base y un desplazamiento (offset). Esto permite que el proceso sea incluso reubicado en la memoria mientras está siendo ejecutado sin tener que sufrir cambios, pero requiere de hardware específico (como una MMU).

**Carga dinámica y enlace dinamico?**

**Overlays (superposiciones)**

Es una técnica (manual, implementada por el programador) en la que solo se conservan en memoria aquellas instrucciones y datos que se requieren en un momento determinado. Cuando se necesitan otras instrucciones, se sobrescriben con las que ya no se necesitan. De esta forma, un proceso puede ser mayor que la cantidad de memoria que se le asigna.

**Ventajas:**

1. No requiere apoyo especial del S.O.. Lo único que nota el sistema operativo es que hay más E/S que de costumbre.
2. La dimensión de un proceso no se limita por el tamaño de la memoria física.

**Desventaja:**

1. El programador debe diseñar y programar adecuadamente la estructura de los overlays. Esta puede ser una tarea de gran magnitud que requiere un conocimiento total de la estructura del programa, su código y sus estructuras de datos.
2. El uso de los overlays está limitado a microcomputadores y otros sistemas con cantidad de memoria física limitada y que carecen de apoyo de hardware para técnicas más avanzadas. Son preferibles las técnicas automáticas para ejecutar programas de gran tamaño en cantidades limitadas de memoria física.

**Direcciones físicas y direcciones lógicas**

* **Direcciones físicas:** hacen referencia a la dirección física real de memoria central. Es la posición en la que se encuentra efectivamente un dato.
* **Direcciones lógicas:** son las direcciones que un proceso maneja. Estas direcciones pueden no coincidir con las físicas, y en tal caso, es necesario traducirlas a las direcciones físicas para poder hallar el dato.

Esta consideración de direcciones físicas y lógicas es importante tenerlas en cuenta en todas las técnicas y todos los algoritmos de planificación y administración de la memoria.

La correspondencia en el tiempo de ejecución de direcciones físicas y lógicas, se logra por la

Unidad de Administración de Memoria (M.M.U. Memory Manager Unit), el cual es un dispositivo de hardware incorporado en el procesador, que posterga los enlaces hasta la ejecución.

Un programa de usuario nunca ve las direcciones físicas reales, trata con direcciones lógicas, el hardware de correspondencia de memoria convierte las direcciones lógicas en físicas. La localidad final de la dirección no se determina hasta que no se efectúe la referencia.

**Técnicas de administración sin swapping**

Son técnicas que nos ayudarán a gestionar y organizar la memoria. Se caracterizan por no hacer uso del swapping.

**def. Swapping:** es el intercambio producido entre dos niveles de almacenamiento. Por ejemplo: la memoria central y el almacenamiento secundario en un disco rígido

* **Memoria dedicada (máquina desnuda sin S.O):** el programador tiene a su disposición toda la memoria. No hay un sistema operativo que sea intermediario. Es el programador el que tiene que encargarse de manejar la memoria, de traducir el código binario, de manejar los recursos.  
    
  Este modelo ofrece mucha flexibilidad, simplicidad, y un mínimo costo. Pero perdemos las ventajas de un S.O (atención de errores e interrupciones), y también se nota una pérdida de productividad para el programador (es más difícil programar, ni hablar de hacer programas largos). La multiprogramación no existe en este modelo.
* **Asignación contigua simple:** la memoria se divide en tres campos contiguos: el primer campo almacena al S.O, el segundo almacena al programa en ejecución, y el tercero es espacio asignado (al proceso) sin utilizar.

Se tienen mecanismos de hardware primitivos que aseguran que un proceso no pueda entrar en la zona de memoria del S.O.

Esta técnica solo se implementa en sistemas monoprogramados. Sus ventajas son que es algo sencillo de implementar, simple de comprender y utilizar. No se necesita un hardware de protección y direccionamiento tan elaborado y costoso.

Sus desventajas son un mal uso de la memoria, ya que hay zonas que no se emplean en absoluto. No soporta multiprogramación. Espacio de direccionamiento del usuario acotado por el tamaño de la memoria disponible.

* **Asignación particionada:** con la llegada de la multiprogramación, la memoria central pasó a ser un recurso compartido por varios procesos. Debido a esto, la misma debe organizarse de cierta forma para poder delimitar el espacio que tiene asignado cada proceso.  
    
  La asignación particionada divide la memoria en un distintas regiones de memoria contigua, en las cuales solo podrá trabajar un proceso por partición. Para evitar que los procesos accedan a regiones que no deben, se utilizan registros base y registros límites que delimitan las direcciones de memoria a las que pueden acceder.

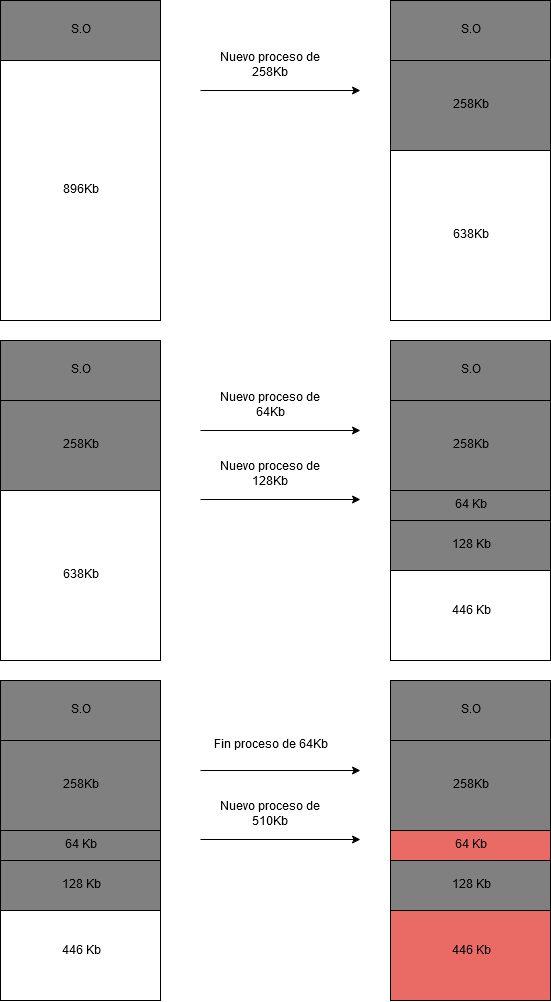
Existen dos tipos de asignación particionada:   
   
**1. Particiones fijas:** la memoria se divide en particiones de tamaño fijo. Cuando un proceso nuevo llega, debe indicar la cantidad máxima de memoria necesaria. A partir de ahí se encuentra una partición acorde y se le asigna al proceso.   
  
El problema que esto genera es que la multiprogramación se ve limitada por la cantidad de particiones que tengamos. Además, pueden generarse casos de fragmentación interna, ya que los procesos pueden requerir menor memoria que el tamaño de la partición, dejando así un espacio sin ser utilizado.

**def. Fragmentación interna:** se denomina fragmentación interna al espacio libre que un proceso no utilizará durante su ejecución, pero que aun así tiene asignado. Este espacio no puede ser aprovechado por otros procesos, ni tampoco será aprovechado por el dueño.



**2. Particiones variables:** la memoria no posee particiones fijas, las mismas se crean al momento de ejecución del proceso, decidiéndose ahí mismo también el tamaño de la misma. Cuando llega un proceso, el mismo indica cuánta memoria requiere, y se crea (si se puede) una partición en memoria de dicho tamaño.  
  
Si bien ahora no tendremos fragmentación interna, produciremos fragmentación externa, debido a que pueden quedar huecos de memoria entre las particiones, cuyo tamaño sea muy chico y no puedan ser utilizados para albergar un nuevo proceso.

Como se ve en la imagen de abajo, hay espacios libres de memoria, pero no pueden ser utilizados para albergar un proceso de tamaño mayor, ya que la memoria está fragmentada externamente.



**def. Fragmentación externa:** es la generación de un gran número de áreas libres, separadas debido al constante intercambio de procesos, entonces se producen huecos muy pequeños en los cuales no cabe ningún proceso. Estas porciones no son aprovechadas por lo que se desperdician y a éste desperdicio se lo llama ***fragmentación externa***. Existe fragmentación externa cuando hay memoria suficiente para un requerimiento, pero no es contigua, sino que está distribuida en pequeñas áreas libres de la memoria central que no es aprovechable.

La fragmentación externa puede solucionarse compactando la memoria. Pero la compactación de memoria es algo muy costoso y que produce mucho overhead, ya que significa mover todos los procesos dentro de memoria, volviendo a tener que calcular las direcciones internas de los procesos.

Para evitar la fragmentación externa en la técnica de particiones variables, se utilizan algoritmos de asignación:

* **Primer ajuste** o ***First-fit*** (**FF**): Recorre la tabla de áreas libres y asigna la primera área donde quepa el requerimiento. La búsqueda puede comenzar en el inicio del conjunto de áreas o a partir de donde terminó la búsqueda del primer hueco anterior.
* **Mejor ajuste** o ***Best-fit*** (**BF**): Utiliza Estrategia Global. Busca el área más chica que pueda contener el requerimiento. Lo mejor es tener la tabla de espacios libres ordenada por tamaño (de otra manera se debe recorrer toda la tabla). Si existe una región exacta del tamaño buscado le es asignada al programa, lo cual no es necesariamente cierto en first fit. Evita la separación de áreas grandes y la fragmentación. Produce áreas libres muy pequeñas.
* **Peor ajuste** o ***Worst -fit*** (**WF**): Asigna el bloque más grande posible. Produce áreas libres más grandes que los otros dos. Es el esquema complementario del BF. Se debe buscar en toda la tabla, a menos que esté ordenada por tamaño. Al producir el área sobrante más grande ésta puede ser más útil que un área más pequeña.
* **Ajuste rápido** o ***Next Fit* (NF):** Es la más utilizada. Estrategia local. Es igual al BF con un puntero de localización. Se tiene una tabla con ***n*** entradas, donde la primera sería un puntero a la cabeza de una lista de huecos de un tamaño dado, la segunda a otra lista de otro tamaño y así sucesivamente.

Empíricamente, se demostró que FF y BF son algoritmos mejores que WF en términos de tiempo y aprovechamiento de la memoria y, en general, FF es más rápida que BF y NF es más rápido que BF y aprovecha mejor la Memoria Central que FF.

**Ventas y desventajas de las particiones  
  
Ventajas:**

1. Mayor grado de multiprogramación y los algoritmos son simples y fáciles de implementar.

2. Hardware poco costoso.

**Desventajas:**

1. El Hardware de direccionamiento y reubicación hace más lento al sistema.

2. Hay un desperdicio de memoria central por fragmentación externa.

3. Se producen tiempos de compactación prolongados.

4. Se necesita conocer la longitud del programa a priori para la asignación.

5. Más memoria utilizada por el Sistema Operativo (Las Tablas ocupan espacio de memoria).

6. Limita el tamaño del programa a ejecutar al máximo tamaño de memoria central.

* **Paginación pura:** la paginación pura es una técnica de gestión que permite asignar memoria de forma no contigua. Para lograr esto, lo que hace la paginación es dividir tanto la memoria física, como la memoria que tiene asignada un proceso.

Esta división es lograda a través de las **páginas** y de los **frames.**

**Frames:** los frames son bloques de memoria física. La paginación divide a toda la memoria física en bloques de igual tamaño.

**Páginas:** las páginas son bloques de memoria de igual tamaño que los frames, que están asignadas a un proceso. Son una división lógica de la memoria.

Lo bueno de esta técnica, es que no es necesario que las páginas estén ubicadas de forma contigua en la memoria física. Un proceso puede tener asignadas páginas que pueden estar en cualquier lado de la memoria. Lo único que es contiguo son las direcciones dentro de una misma página.

Para poder realizar todo esto se necesitan tener dos tablas: una tabla de páginas, y una tabla de frames.

**Tabla de páginas (MPT Memory Page Table):** Uno por cada proceso activo. Describe los bloques de Memoria Central que están siendo usados por el proceso (es decir, en qué frame está cada página). Contiene una entrada por cada página del proceso. Existe una por cada proceso cargado en el sistema.

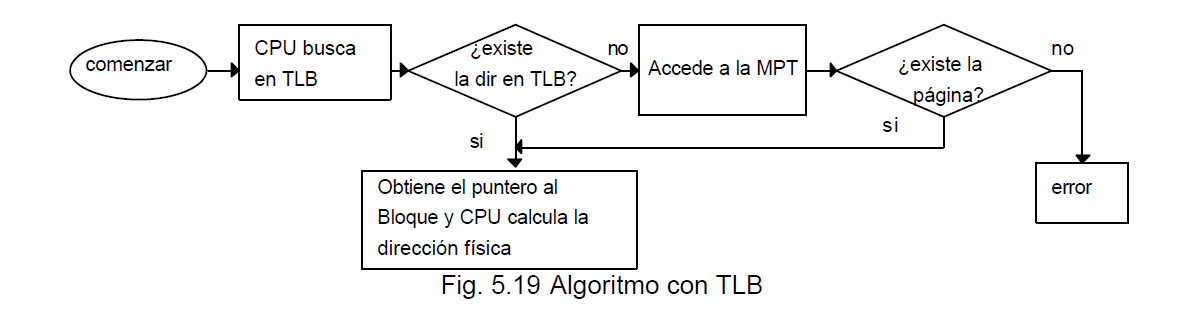
**Tabla de frames (MFT Memory Frame Table):** Existe una sola por sistema. Describe el estado de cada frame de la memoria central (áreas libres para asignar y las ya asignadas, o sea ocupadas). Esto se puede resolver con un solo bit para indicar el estado de LIBRE o ASIGNADO. Lleva la contabilidad.

El tamaño de página debe ser elegido con cuidado, ya que, a menor tamaño, mayor cantidad de entradas en la tabla. Si es muy grande, hay fragmentación pues los procesos no ocupan la totalidad del tamaño de la página y en algunos casos solo ocupan unos pocos bytes. Si bien son deseables los tamaños de páginas pequeños, cada entrada en la tabla de páginas implica bastante gasto adicional, como ser recursos ocupados, tiempo de búsquedas, etc. que disminuye al aumentar el tamaño de la página.

Además, a pesar de eliminarse la fragmentación externa, la fragmentación interna aún existe, y se da en la última página que tiene un proceso.

**TLB (translation lookaside buffer):** el TLB es un buffer de alta velocidad usado para mejorar el rendimiento de la paginación. Su funcionamiento es similar a la caché de memoria central. Lo que hace es guardar las páginas recientemente usadas en conjunto con el frame que referencian. De esta forma se omite el paso de ir a la tabla de páginas del proceso y realizar la traducción de página a frame.

Una característica interesante de la TLB es que sus registros son asociativos. Es decir, es como si fueran un HashMap donde el número de la página es la clave (key), y el valor (value) es el frame asociado con esa página. El procesador está equipado con un hardware especial que chequea en un mismo instante todas las entradas de la TLB en busca de un acierto.



**Páginas compartidas**

Sólo se pueden compartir páginas cuando dicha página está programada como código reentrante (o puro, que no se modifica a sí mismo). Nunca cambia durante la ejecución, de manera que dos o más procesos pueden ejecutar el mismo código al mismo tiempo. Consiste en apuntar las tablas de páginas de todos los procesos a los mismos frames.

**Ventajas:**

1. Especial importancia en un entorno de tiempo compartido.

2. Ahorro considerable del espacio de memoria.

3. Se pueden compartir programas de uso frecuente: compiladores, sistemas de bases de datos, etc.

**Desventajas:**

1. El S.O. deberá asegurar la propiedad de sólo lectura del código compartido.

**Protección**

* Se usan Bits de protección asociados con cada frame (R, W, X). Normalmente estos bits se conservan en la tabla de páginas. Al mismo tiempo que se calcula la dirección física, pueden consultarse los bits de protección. Por ejemplo, un intento de escribir en una página de sólo lectura ocasiona una interrupción al S.O. (violación de protección de memoria).
* Un bit es adherido a cada entrada de la tabla de página: bit válido - inválido. El S.O. pone este bit para cada página, para permitir o no el acceso a la página. Cuando el bit es “válido” este valor indica que la página asociada está en el espacio de direcciones lógicas del proceso, y es entonces una página legal. Si es “inválido” no está en el espacio de direcciones lógicas y es ilegal. (válido puede ser 1 e inválidos = 0)
* Es poco común que un proceso utilice todo su intervalo de direcciones, por lo tanto, es un derroche crear una tabla con registros para cada página del intervalo de direcciones. La mayor parte de esta tabla no se utilizaría, pero ocuparía un valioso lugar en memoria. Algunos sistemas ofrecen hardware que indica el tamaño de la tabla de página a través de un Page Table Length Register (PTLR) para que el proceso no pueda direccionar más allá de un cierto límite.

**Bit de Modificación de página**

Como algunas de las páginas en memoria central sufren alteraciones durante la ejecución (ejemplo: aquellas con datos). Cuando finaliza la ejecución o cuando se requiera, deben actualizarse en el disco éstas modificaciones, por lo que se necesita contar con un bit que indique cuál fue la página alterada para seleccionarla y grabarla en el disco.

**Paginación multinivel**

Es un esquema donde existe una jerarquía de tablas de páginas. Este esquema surge cuando el tamaño de la tabla de páginas se vuelve muy grande y no puede ser almacenado en un solo frame de memoria. El tamaño de las tablas permanece igual, pero tenemos más tablas, pudiendo así almacenar más páginas para un mismo proceso.

**Ventajas y desventajas de la paginación**

**Ventajas**  
1. Elimina el problema de fragmentación externa.

2. Mayor grado de multiprogramación.

3. No necesita compactación.

4. Permite que la memoria de un proceso no sea contigua, y que a un proceso se le asigne memoria física donde quiera que ésta esté disponible. Evita el gran problema de acomodar trozos de memoria de tamaño variable en el almacenamiento auxiliar.

**Desventajas**

1. Existe Fragmentación interna.

2. Hardware más caro.

3. Menor velocidad debido a la constante traducción de direcciones lógicas en físicas.

4. Gasta más memoria central utilizado con tablas.

5. El PCB de cada proceso crece, debe almacenar los contenidos del espacio de direccionamiento.

6. Espacio de memoria de un proceso debe ser menor o igual que el de memoria central libre para alojarlo (esta es una importante restricción)

* **Segmentación simple:** un programa de usuario puede ser subdividido usando segmentación, lo que significa que el programa y sus datos asociados son divididos en un número de segmentos. No es un requerimiento que todos los segmentos del programa sean del mismo tamaño, aunque existe un tamaño máximo de segmento.   
    
  Como pasa en la paginación, la dirección lógica de un segmento consiste en dos partes, en este caso un número de segmento y un offset.   
    
  Debido al uso de segmentos de diferentes tamaños, la segmentación es similar a las particiones variables. En la ausencia de un esquema de overlays (superposición) o memoria virtual, se requeriría que todos los segmentos de un programa sean cargados en memoria para su ejecución. La diferencia, comparado con las particiones variables, es que con la segmentación un programa puede ocupar más de una partición, y estas particiones no necesariamente tienen que ser contiguas.

La segmentación elimina la fragmentación interna, pero igual que en las particiones variables, sufre de fragmentación externa. Sin embargo, como un proceso es dividido en pequeñas partes, la fragmentación externa debería ser menor.   
  
A diferencia de la paginación, la segmentación es visible al programador y es generalmente una conveniencia para organizar al programa y sus datos. Típicamente, el programador o el compilador asignará programas y datos a diferentes segmentos.

Otra consecuencia de usar segmentos de tamaños diferentes es que no hay una relación simple entre la dirección lógica y la dirección física. Análogo a la paginación, un esquema de segmentación simple haría uso de una tabla de segmentos (Memory Segment Table) para cada proceso y una lista de bloques libres de memoria central. Cada registro de la tabla de segmentos debería indicar la posición de inicio del segmento en memoria central. El registro también debería proveer el tamaño del segmento, para asegurarse de que direcciones inválidas no son usadas. Cuando un proceso entra en estado de ejecución, la dirección de su tabla de segmentos es cargada en un registro especial usado por la MMU.

**Segmentos compartidos**

Otra ventaja de la segmentación involucra el compartir códigos o datos. Cada proceso tiene una tabla de segmento asociada con su PCB, a quien el dispatcher usa para definir los contenidos de la tabla de segmento cuando a éste proceso se le da la CPU. Los segmentos son compartidos cuando las entradas en las tablas de segmento de dos procesos diferentes apuntan a las mismas locaciones físicas.

El compartir ocurre a nivel de segmentos. Así, cualquier información puede ser compartida si ésta se definió previamente para ese segmento.

Por lo dicho, se puede compartir código fácilmente. Debido al tipo de direccionamiento, los segmentos compartidos deben tener el mismo número o nombre en los SMT de todos los procesos que los usan (Por ejemplo: Bibliotecas). Pueden compartirse varios segmentos, por lo que un programa compuesto por varios segmentos también puede compartirse total o parcialmente (Por ejemplo: editor de texto en un sistema de tiempo compartido).

Se presentan ciertas dificultades: los segmentos de código con frecuencia contienen referencias a sí mismos. Por ejemplo, un salto condicional normalmente tiene una dirección de transferencia, la cual se compone de un número de segmento y un desplazamiento. El número de segmento de la dirección de transferencia será el mismo número del segmento de código. Si tratamos de compartir este segmento, todos los procesos que lo comparten deben definir el segmento de código compartido con el mismo número de segmento.

Los segmentos de sólo lectura pueden compartirse con números de segmentos distintos, al igual que muchos segmentos de códigos que no se refieren a sí mismos de manera directa, sólo directa.

**Protección de Segmentos compartidos**

Una ventaja particular de la segmentación es la asociación de la protección de los segmentos. Ya que los segmentos representan una porción definida semánticamente del programa, es probable que todas las entradas en el segmento sean usadas de la misma forma. De ahí que se tienen segmentos que son instrucciones mientras que otros segmentos son de datos. El hardware de mapeo de memoria (M.M.U.) verificará los bits de protección asociados con cada entrada a la tabla de segmentos para prevenir accesos ilegales a memoria.

Localizado un array en su propio segmento, el hardware de administración de memoria verificará qué índices del array son legales y cuales no (están fuera de los límites de arreglo). Así, los errores comunes de programas serán detectados por el Hardware antes de que causen serios daños.

Los segmentos se pueden proteger (contra Lecturas, escrituras o ejecución) y especificar su tipo (R, W, X, etc.) agregando bits a la SMT. Esto es muy caro por lo que se propone agregar un puntero a una Tabla de Protección, en donde cada entrada tiene asociado los procesos y sus permisos.

Las referencias fuera de segmento se pueden tomar como error o como necesidad de más memoria y así, disparar un mecanismo de crecimiento dinámico de segmentos.

**Bit de modificación de un segmento**

Como en la paginación, la segmentación requiere de más bits adicionales en la SMT para indicar si el segmento fue modificado. Si el bit fue activado significará que ese segmento debe ser grabado en el disco en algún momento de su ejecución o al finalizar para mantener los datos actualizados.

**Implicaciones de las técnicas de administración sin swapping**

Las técnicas como la paginación y la segmentación dan dos características notables:

* Las referencias a memoria de un proceso son direcciones lógicas que pueden ser traducidas direcciones físicas. Esto significa que un proceso puede intercambiarse dentro y fuera de la memoria central, y que puede ocupar espacios diferentes de la misma.
* Un proceso puede descomponerse en varios objetos (páginas o segmentos), y estos objetos no necesitan estar todo el rato cargados en memoria central durante la ejecución.

Esto, en conjunto con el swapping, da a luz a dos implicaciones:

1. Más procesos pueden mantenerse en la memoria central. Porque sólo vamos a cargar algunas de las partes de cualquier proceso particular. Esto lleva a la utilización más eficaz del procesador, ya que es más probable que por lo menos uno de los procesos esté en estado Listo en cualquier momento en particular.

2. Es posible para un proceso ser más grande que toda la memoria central. Una de las

restricciones más fundamentales de la programación de alto nivel. Sin el esquema, podemos darnos cuenta de que un programador debe ser agudamente consciente de cuánta memoria está disponible. Si el programa escrito es demasiado grande, el programador debe inventar maneras de estructurar el programa en pedazos (objetos) para que puedan cargarse separadamente en alguna clase de estrategia de overlay.

De estas implicaciones nace el concepto de **memoria virtual**. La memoria virtual es una técnica que se encarga de que el sistema operativo disponga, tanto para el software de usuario como para sí mismo, de mayor cantidad de memoria que esté disponible físicamente.

El enfoque de memoria virtual utiliza el S.O. para producir la ilusión de una memoria sumamente grande, que se denomina memoria virtual**.**

Existen dos técnicas de memoria virtual principales: la administración de memoria paginada bajo demanda o solicitud y la administración de memoria mixta: segmentada con paginación por demanda.

**Técnicas de administración con Swapping**

Hasta ahora, con las administraciones anteriores, no era posible correr un trabajo hasta que

hubiera memoria disponible suficiente para cargar todo su espacio de direccionamiento que necesita ese trabajo para su ejecución.

Esta restricción es importante, ya que los programas, cuyos espacios de direccionamiento son

mayores que la existencia de áreas libres no utilizadas no es posible cargarlos ni ejecutarlos.

Se define **swapping** como a la técnica de intercambio entre dos niveles de memorias.

A la operación de enviar o grabar la información al disco se le llama **swap out**, y la de cargar o leer la información desde disco a memoria se le llama **swap in**.

**Restricciones del Swapping:**

El swapping tiene serias desventajas en lo que a velocidad del sistema se refiere, ya que la

velocidad de transferencia y tiempo de acceso de los discos es varias veces mayor que la de memoria central, por lo cual sólo se deben realizar intercambios cuando sea necesario, es decir, cuando la memoria central no sea suficiente. Los procesos que se intercambian deben estar completamente inactivos, no se puede intercambiar procesos en espera de entrada salida, porque los datos podrían perderse, teniendo que terminar el proceso. Entonces:

* Nunca intercambiar un proceso con E/S pendiente. El proceso debe estar completamente inactivo (Bloqueado).
* Ejecutar las operaciones de E/S solo en los buffers del S.O.

Al exceso de intercambios se le llama ***trashing***, porque es un desperdicio del tiempo de proceso.

**Paginación bajo demanda o bajo solicitud**

Esta técnica no requiere que se cargue todo el programa y sus datos en la memoria central como lo requería la paginación pura.

El principio de localidad establece que la mayor parte del tiempo de ejecución del programa

transcurre en una porción reducida de su código, por lo que cargar todo el programa en memoria puede ser un desperdicio. Algo similar ocurre con los datos, la mayoría de los programas no usan todos sus datos todo el tiempo. Con la memoria virtual estos inconvenientes se solucionan, cargando en memoria lo que se necesita y enviando a disco lo que no se necesita.

La administración de memoria paginada por demanda elimina el requerimiento que todo

el espacio de dirección de un trabajo está en la memoria central al mismo tiempo; en lugar de ello solo se cargan unas pocas porciones del mismo.

Las razones fundamentales por lo que los programas usan una pequeña porción de todo su espacio de direccionamiento durante una ejecución son que aquellas porciones de programa que se ejecutan deben estar cargadas en la memoria central y las que no se ejecutan pueden estar en el disco, como por ejemplo las rutinas de manejo de errores que generalmente se invocan al ocurrir el problema, entonces se cargan cuando se las necesita. Otro motivo puede ser que determinadas rutinas de los programas se usan en tiempos mutuamente excluyentes por ejemplo en un ABM (Alta, Baja o Modificación) de Clientes, cuando se procesa un alta no se procesa una baja o una modificación, etc.

Cuando se planifica inicialmente un trabajo para su ejecución, generalmente solo se carga

físicamente sus primeras páginas, y las demás páginas que necesite el trabajo se cargan

subsecuentemente bajo solicitud o demanda de la ejecución, por lo que la memoria se asigna y se desasigna durante la ejecución del trabajo.

Cuando un trabajo hace referencia a un área del espacio de direccionamiento que no está en la

memoria física, el Hardware de mapeo genera una ***interrupción por falta de página (Page fault).*** El S.O. procesa ésta interrupción, no como una excepción (trap) sino una interrupción que realiza el servicio de cargar las páginas requeridas desde el almacenamiento secundario (swap-in) y actualiza en forma adecuada las entradas de la tabla de páginas, luego reinicia la ejecución de la instrucción interrumpida.

Las páginas solicitadas están cargadas en un dispositivo de almacenamiento secundario donde se ha reservado una copia de todo el espacio de direccionamiento del trabajo.

Una vez que la memoria se ha llenado con páginas y ante la demanda de un espacio adicional,

solo es posible cargar una serie de páginas nuevas desde el disco si previamente se elimina primero algunas de las que ya estén residentes en la memoria.

Las páginas reemplazadas (víctimas) se copian al dispositivo de almacenamiento secundario antes de cargar la nueva si es que sufrieron alguna modificación durante su procesamiento. Esta técnica se llama ***sustitución, intercambio o trueque de página***.

**Tipo de sustitución de páginas**

**Observación sobre el bloqueo de frames**: Cuando un frame está bloqueado, la página cargada actualmente en este frame no puede ser reemplazada. La mayoría de los Kernels de los sistemas operativos, así como las estructuras clave de control, se albergan en frames bloqueados.

**Sustitución global (Asignación dinámica de frames)**

* El proceso puede tomar un frame a sustituir de cualquier lugar de memoria central, aunque el frame esté asociado a otro proceso.
* Puede suceder que la cantidad de frames de un proceso crezcan o disminuyan. Es posible que un proceso seleccione solamente frames de otros procesos, incrementando el número de frames que tiene asignado (suponiendo que otros procesos no elijan sus frames para reemplazo).
* Los procesos no son responsables ni pueden controlar su propio page fault rate.
* El conjunto de páginas en memoria para un proceso no depende solamente del comportamiento de ese proceso, sino también del de otros.

**Sustitución local (Asignación estática de frames)**

* El proceso debe tomar un frame de su propio conjunto de frames asignados a él.
* El número de frames asociados al proceso no cambia.
* El proceso sólo está afectado por su propio paginado.
* Puede desaprovechar frames no usados por otros procesos o afectar a un proceso al no permitir que tenga acceso a otras páginas menos utilizadas.

Generalmente es mejor el reemplazo global (es más productivo).

**Algoritmos de sustitución de páginas**

* **Óptimo:** se reemplaza la página que no va a ser usada por el máximo periodo de tiempo. Es el algoritmo que mejor tasa de fallas tiene. Pero a su vez, es costoso de implementar, y necesita conocimiento a futuro, lo cual es casi imposible de conseguir.
* **FIFO:** se reemplaza aquella página que más tiempo ha permanecido en memoria. Puede usarse una cola FIFO para saber qué página es la más vieja.

**Ventajas:**

1. Elimina la posibilidad de cargar una página e inmediatamente sustituirla. Si una página se usa mucho, y durante un largo período de tiempo, será la mejor candidata a ser sustituida.

2. Es fácil de comprender y programar.

3. La página reemplazada puede ser un módulo de asignación de valores iniciales que se utilizó hace mucho tiempo y que ya no se necesita.

**Desventajas:**

1. Presenta la llamada **Anomalía de Belady**: bajo ciertas trazas, el page fault ratio crece cuando se agregan más frames a la memoria central, o dicho de otro modo, dada una traza, cuánto más frames se asignan al proceso, más fallos de páginas produce.

2. La página reemplazada puede contener una variable cuyo valor inicial se asignó hace tiempo pero que se utiliza constantemente.

* **LRU (Least Recently Used):** se reemplaza la página que menos se ha usado en durante un período de tiempo.   
    
  **Ventajas:**

1. No sufre la anomalía de Belady.

2. Se puede demostrar que el conjunto de páginas en memoria para n frames es igual al conjunto de páginas para n + 1 frames. Ya que con n frames tendríamos en memoria central las n páginas más usadas recientemente y con n + 1 frames pasaría lo mismo. Esto no era cierto con FIFO.

**Desventajas:**

Se necesita llevar un registro del uso de las páginas y esto lo debe hacer el Hardware a través de dos formas de llevar esta cuenta. El hardware implementa solo una de estas técnicas: Hora de uso, o Pila.

En hora de uso se utiliza un contador que es incrementado por la CPU. La página que tenga menos tiempo de uso es reemplazada.

La técnica de pila utiliza una pila donde se guardan las páginas. La página más reciente es guardada al tope de la pila, la menos reciente está al fondo, y esta es el candidato a ser reemplazada.

Todo esto genera mucho overhead.

* **Second chance:** FIFO modificado con un reference bit. Si una página es seleccionada como víctima, primero se verifica su reference bit. Si es 0, se la reemplaza. Si es 1 se le da una segunda oportunidad, se limpia su reference bit y se selecciona otra página en orden FIFO.

A la página que se le brinda una segunda oportunidad se le establece como momento de llegada el momento actual. De esta manera, esta página no se va a reemplazar hasta que todas las demás páginas sean reemplazadas. Si una página se utiliza con frecuencia suficiente para mantener en 1 su bit de referencia, nunca será reemplazada.

**Hay otros algoritmos más. Recomiendo ver el notas, porque básicamente no estoy resumiendo nada acá.**

**Sistemas mixtos: segmentación con paginación por demanda**

* Combinan las técnicas de Segmentación y de paginación.
* En general los segmentos tienen un tamaño múltiplo de páginas.
* No se necesitan tener cargadas en memoria central todas las Páginas de los segmentos.

La dirección virtual se organiza en tres partes:



La traducción se realiza de la siguiente forma:

* Primero: Se efectúa una búsqueda asociativa en la tabla de segmento (SMT) y luego en la tabla de página (MPT) de ese segmento. Puede ocurrir:

1. Que el segmento no está cargado (no existe el bit de presencia de segmentos en memoria central), entonces se produce una interrupción por falta de segmento en memoria central (***Segment fault***), luego se genera una búsqueda del segmento en el Disco, se crea la tabla SMT y la MPT para ese segmento. Si hay lugar se cargan las páginas necesarias y se reinicia le ejecución.

2. Que la página referenciada no está cargada en memoria central por lo que se produce una interrupción de falla de página (***page fault***).

* Segundo: una vez localizada la correspondencia entre segmento y página en las tablas y hecho la suma del desplazamiento, se obtiene la dirección real donde está cargada en la memoria física.

El mecanismo de traducción dinámica requiere de Registros rápidos para una operación eficiente (valen los mismos considerandos ya explicados en Paginación por Demanda) ya que se requiere tres ciclos: uno para acceder a la tabla de segmentos (SMT), otro para la tabla de páginas (MPT) y el tercero para el direccionamiento final.

**Ventajas:**

1. Permite compartir segmentos.

2. No es necesario cargar la totalidad de los segmentos en memoria central ni la totalidad de las

Páginas. Solo lo que se necesite.

3. No se requiere compactación.

**Desventajas:**

1. Requiere más Hardware para el direccionamiento.

2. Es más lento en la ejecución (por el mecanismo de traducción de las direcciones virtuales)

3. El S.O. ocupa más Memoria.

4. Aumenta la fragmentación interna.

**Administración de E/S**

**Introducción**

Las razones del porqué los periféricos no se conectan en forma directa al bus del sistema son las siguientes:

* Hay una amplia variedad de periféricos con varios métodos de operación. Sería impráctico incorporar la lógica necesaria dentro del procesador para controlar un rango de dispositivos uno por uno. Además, si apareciera un nuevo tipo de dispositivo, este no se podría conectar.
* La velocidad de transferencia de datos de los periféricos es a menudo bastante menor que la de la memoria central o del procesador. No resulta práctico utilizar un bus de alta velocidad para comunicarse directamente con el dispositivo periférico.
* Los periféricos generalmente emplean distintos formatos de datos y longitudes de palabra que los utilizados en el computador al que se conecta.

En consecuencia, es necesario un módulo de E/S. Este tiene dos funciones principales

desde el punto de vista del Hardware:

* Proveer una interfaz con el procesador y la memoria central vía el bus del sistema.
* Proveer una interfaz a uno o más dispositivos periféricos con vínculos especialmente diseñados.

**El Sistema de Gestión de Entrada / Salida**

El Sistema de Gestión de la E/S es la parte del S.O. que se ocupa de la organización, administración y operación de los dispositivos de E/S.

Entendemos por dispositivos de E/S a los periféricos con su respectiva unidad de control, conectados a los controladores y a los puertos (Ports) o canales que ofician de interfaz entre el bus del sistema y los periféricos. Cada periférico tiene distintos requerimientos. No funcionan de la misma forma.

La gestión de E/S debe ser eficiente, segura, confiable, abstracta, uniforme, etc. y actúa como

interfaz entre el programa en ejecución y los dispositivos, proveyendo una abstracción de las complejas operaciones y controles entre ellos.

Las actividades que realiza el sistema de E/S son concurrentes con el procesador y son autónomas, pero no autosuficientes ya que requieren de la colaboración de otros módulos del S.O. (File System y administrador de memoria central).

Las operaciones de E/S funcionan concurrentemente, dado que se pueden iniciar una serie de

operaciones independientemente de que se completen otras en progreso ya iniciadas. Estas tareas concurrentes requieren de un adecuado planeamiento para que no ocurran deadlocks.

**Funciones del administrador de E/S**

* Mantener un registro de estado de los dispositivos periféricos, controladores y canales y de los programas que controlan a estos dispositivos.
* Determinar una técnica de asignación mediante un procedimiento de vinculación (Binding).
* Controlar todos los Dispositivos de E/S (emitir comandos, capturar interrupciones y manejar errores generados por los dispositivos, inicializar los dispositivos en el arranque y en cada operación, procesar, cancelar y sincronizar las operaciones, etc.)
* Proveer una interfaz entre los dispositivos y el resto de los Sistemas (Lógicamente abstracta, independiente, simple y fácil de usar, la misma para todos los dispositivos, etc.)
* Organizar, administrar y operar la transferencia de los datos y la información desde y a los dispositivos, ya sea temporal o definitivamente en forma eficiente.
* Conjuntamente con el Sistema de Gestión de Archivos, manejar los datos almacenados en los soportes, que implica la conversión de formatos, factores de bloqueos, ediciones parciales, apertura y cierre de archivos, almacenamiento y recuperación de la información. Manejar las colas de impresión (Spooler), etc.
* Proveer un mecanismo de protecciones y control de accesos.

**Módulos de E/S**

**Funciones del módulo**

El módulo de E/S es algo ajeno al S.O. Es una entidad física dentro de la computadora, que tiene la responsabilidad de controlar uno o más dispositivos externos e intercambiar los datos entre esos dispositivos y la memoria central y/o los registros de la CPU. En consecuencia, el módulo de E/S debe tener una interfaz interna al computador (al procesador y la memoria central) de alta velocidad y una interfaz externa al computador (al dispositivo externo).

Las funciones primordiales o requerimientos para un módulo de E/S caen dentro de las siguientes categorías:

* **Control y Temporización:** debido a que los recursos internos de la PC son compartidos por más dispositivos de E/S, se necesita algo que regule el tráfico de datos y las señales entre dispositivos internos y externos
* **Comunicación con el procesador:** se necesita un mediador que se comunique con el procesador y le informe qué debe hacer con el dispositivo de E/S.
* **Comunicación con el dispositivo:** a su vez, el procesador tiene que poder darle comandos al dispositivo para que realice acciones.
* **Amortiguación (buffering) de datos:** las velocidades de transferencias entre el dispositivo de E/S y el procesador (y la memoria central) difieren bastante. Para evitar ralentizar todo el sistema, tiene que haber un buffering de datos que haga más eficientes las transferencias.
* **Detección de errores:** el módulo de E/S tiene que poder detectar cuando un dispositivo externo está fallando, tanto físicamente, o también en el envío de datos (bits de paridad, Hamilton, etc.).
* **Manejo de Interrupciones en el bajo nivel:** el módulo debe ser capaz de manejar y atender las interrupciones de los dispositivos de E/S.

**Operaciones de hardware de E/S**

**Operación asincrónica**

Los dispositivos de E/S funcionan independientemente del procesador y del clock interno del mismo. No podemos predecir cuándo van a llegarnos datos de un dispositivo de E/S, además, las acciones que realizan los dispositivos de E/S dependen de su propio funcionamiento o de lo que dicte el usuario (el teclado solo manda datos cuando yo tipeo, por ejemplo).

Esto hace que las operaciones de los dispositivos sean totalmente asincrónicas. Debido a esto deben establecerse protocolos de comunicación en los que quede claramente establecido cuándo un dispositivo está por empezar a transmitir datos, cuál es el estado del dispositivo, etc.

**Diferentes velocidades**

Los dispositivos de E/S son un Fiat 147. El procesador es un Tesla Modelo S tuneado y con subwoofers para pasar cumbia a todo lo que dé.

Las velocidades que se manejan son de órdenes de magnitudes diferentes. Se necesitan técnicas para contrarrestar estas diferencias y hacer que el funcionamiento sea rápido y eficiente. El buffering es una de dichas técnicas. También lo es el acceso directo a memoria (DMA).

**Controladores (driver)**

Los dispositivos de E/S necesitan poseer un controlador. El controlador es lo que genera la interfaz entre el dispositivo y el S.O. Es la capa de abstracción que le permite al programador ver los dispositivos como si fueran una caja negra, sin tener que preocuparse de su funcionamiento interno. Un controlador puede controlar a una o más unidades de periféricos al mismo tiempo.

Una de las funciones del controlador en convertir un flujo de bits a un bloque de bytes y realizar las acciones de corrección de errores necesarias.

Un controlador básicamente está compuesto por tres partes:

1. **Una interfaz con el bus del sistema:** son los zócalos de conexión, los puertos donde conectamos nuestros dispositivos de E/S. Los controladores necesitan saber cómo comunicarse a través de ese puerto, cómo utilizarlo para comunicarse con el sistema.
2. **Un controlador propiamente dicho (Lógica programada):** es la parte que hace de interfaz entre el S.O y el dispositivo, provee la separación lógica/física, y la abstracción de las operaciones de E/S.
3. **Una interfaz con el dispositivo (Uno o más):** adapta el dispositivo al sistema, provocando que el mismo sea dependiente del controlador por los niveles de señales y las órdenes que lo hacen funcionar.

**Puertos**

Un puerto es una interfaz a través de la cual se pueden enviar y recibir los diferentes tipos de datos. Es una abstracción de los registros (bufferes) que se encuentran dentro del dispositivo de E/S.

Las Direcciones de los puertos pueden estar implementadas en dos formas:

1. En el espacio de direcciones de Memoria Central asignado a cada controlador (mapeados en memoria), por lo que los puertos pueden ser accedido y operado como si fuera la Memoria Central.
2. En un espacio de direcciones separados (mapeados en E/S o E/S aislada). Los accesos a los puertos deben ser realizados por medio de instrucciones hardware especiales que activan las señales de direccionamiento del espacio de E/S (ejemplo ent\_port (in) o sal\_port (out)).

La primera es más eficiente, mientras la segunda requiere efectuar copias de direcciones en los registros internos del procesador (acción que recarga el procesamiento de E/S pero es más simple de implementar y no ocupa espacio de memoria central por lo que es un método muy utilizado).

**Discos**

**Acá el Notas habla sobre los distintos tipos de discos que existen (HDD, CD, DVD, ED, EDD, y EDDY), cómo funcionan, y demás. No voy a resumir eso, es un embole resumirlo.**

**Parámetros de rendimiento de discos**

Los discos duros tienen una serie de parámetros qué indican cuánto tiempo se demoran en realizar una determinada acción. Los mismos son:

* **Tiempo de búsqueda:** es el tiempo de búsqueda necesario para mover el brazo del disco hasta la pista solicitada.
* **Tiempo de latencia:** es el tiempo de espera necesario hasta que el sector solicitado pasa por debajo del cabezal.
* **Tiempo de transferencia:** tiempo en el que tarda en cargar un dato a memoria.
* **Tiempo de acceso:** es la suma de los anteriores tiempos.

**Algoritmos de Planificación del Brazo del Disco**

Cuando la unidad de disco está operando, el disco gira a una velocidad constante. Para leer o

escribir la cabeza debe posicionarse en la pista al comienzo del sector pertinente. Si el sistema es de cabezas móviles hay que mover la cabeza para elegir la pista, y si es de cabezas fijas habrá que seleccionar electrónicamente una de ellas.

El tiempo para leer o escribir un bloque en el disco, decíamos que, básicamente está determinado por estos tres factores: el tiempo de búsqueda (movimiento del brazo al cilindro adecuado), la demora rotacional (hasta que pase el sector buscado por debajo de la cabeza) y el tiempo de transferencia además depende de la organización del archivo.

El más largo es el tiempo de búsqueda. Si los pedidos se aceptan de a uno y se realizan de a uno, no se puede hacer nada para optimizar el tiempo de búsqueda. Sin embargo, si mientras se realiza una operación se pueden registrar los pedidos surgidos es posible una mejora.

Algunas unidades mantienen una tabla, ordenada por el número de cilindro, con todos los pedidos pendientes para cada cilindro juntos en una lista enlazada encabezada por las entradas a la tabla.

Los algoritmos de planificación de disco dictan cómo los datos serán buscados los datos (cómo se moverá el brazo del disco). Son muy importantes ya que un algoritmo de búsqueda poco eficiente, reducirá toda la performance del sistema (ya que, si usamos paginación y swapping, será mucho más lento volver a cargar un proceso en memoria, o páginas del mismo).

**FIFO (FCFS)**

La manera más sencilla de planificación es FIFO, que tiene la ventaja de ser justa porque las

peticiones son servidas en el orden de llegada. Sin embargo, esta política suele ser a menudo poco eficiente y puede no proveer el mejor servicio (promedio).

El problema con este algoritmo surge cuando tenemos que pasar a buscar un dato que se encuentra en una pista muy lejana. El brazo va a perder mucho tiempo ubicándose en esa pista. Si tenemos algo como 120 🡪 10 🡪 130 el tiempo gastado en ubicar el brazo es descomunal.

Este algoritmo es muy útil para planificar discos con poca carga (pocos accesos) y es el más simple de programar, pero tiene mucho thrashing.

**def. Thrashing:** situación en la que se utiliza una creciente cantidad de recursos para hacer una cantidad de trabajo cada vez menor.

**STSF (Shortest Time Seek First)**

La política de SSTFconsiste en elegir la solicitud de E/S que requiera el menor movimiento posible del brazo del disco desde su posición actual. Parece razonable servir juntos todos los requerimientos (peticiones) cercanos a la actual posición de la cabeza, antes de mover la cabeza para servir otro requerimiento lejano.

El algoritmo SSTF selecciona el requerimiento con un mínimo tiempo de búsqueda desde la actual posición de la cabeza. Puesto que el tiempo de búsqueda es generalmente proporcional a la diferencia de pistas entre los requerimientos, implementaremos este método por movimiento de la cabeza a la pista más cercana en la cola de requerimientos.

Este algoritmo resultaría en una importante mejora en el servicio de disco promedio en comparación con FIFO.

Esta política puede generar inanición, ya que puede llegar una posición que nunca sea atendida por permanecer lejos en relación a otras peticiones.

**SCAN**

El brazo sólo se mueve en un sentido resolviendo todas las peticiones pendientes de su ruta hasta que alcance la última pista, entonces cambia la dirección de servicio y el rastreo sigue en sentido opuesto. La cabeza continuamente se mueve sobre el disco de extremo a extremo.

Esto genera mucho tiempo de espera para los pedidos detrás del cabezal, ya que no serán atendidos hasta que el mismo cambie de sentido. Sin embargo, no produce la inanición que producía el algoritmo STFT.

**C-SCAN**

Una mejora al SCAN es la política C-SCAN que recorre el disco en una sola dirección, cuando se haya visitado la última pista el brazo vuelve al extremo opuesto del disco (sin atenderse ninguna petición en el viaje de vuelta) y comienza a recorrerlo de nuevo. El viaje de vuelta que dura una fracción despreciable de tiempo.

El algoritmo planificador C-SCAN trata esencialmente al disco como si fuera circular, con el último track adyacente al primero.

**LOOK-UP**

Es semejante al SCAN, pero sólo llegan hasta la última petición y no hasta el final del disco o sea su última pista*.*

Las cabezas se mueven en una dirección hasta que terminan los pedidos en esa dirección, entonces cambian la dirección y realizan los otros pedidos. Solo necesita que el software mantenga un bit indicando la dirección y que la cola esté ordenada en forma ascendente (para el movimiento de pista cero a máxima) y en forma descendente (para el movimiento inverso).

Nótese que, como los describimos, tanto el planificador SCAN como el C-SCAN siempre mueven la cabeza desde un extremo del disco hasta el otro, abarcando la totalidad de las pistas en su recorrido. En la práctica, ninguno de los dos algoritmos se implementa de este modo. Se implementan los algoritmos LOOK-UP y C-LOOK-UP, donde la cabeza es sólo movida tan lejos como el último requerimiento en cada dirección. Tan pronto como no hay requerimientos en la actual dirección, el movimiento de la cabeza se invierte.

**C-LOOK-UP**

Siempre atiende los pedidos en un solo sentido, desde el más bajo al número de cilindro del pedido más alto.   
  
Empieza por la primera petición y recorre de forma ascendente hasta la última. Luego, baja bruscamente a la petición que se encuentre en el lado de inicio, y vuelve a subir.

**SCAN de N pasos y FSCAN**

Otras políticas pueden ser utilizando distintas administraciones de colas.

Con políticas como FIFO, SCAN y C-SCAN, es posible que el brazo no se mueva durante un tiempo considerable si uno o más procesos realizan muchos accesos a una misma pista. Para evitar esto se utiliza la política **SCAN de N** pasos o **FSCAN**.

La primera divide la cola de peticiones en subcolas de longitud N y procesa las subcolas una a una mediante un SCAN. La política FSCAN usa dos subcolas, cuando comienza un rastreo todas las peticiones están en una de las colas y la otra está vacía y durante el recorrido las nuevas peticiones se colocarán en la segunda cola.

**¿Cómo elegir un algoritmo de planificación?**

El algoritmo SSTF es común y tiene un interés natural. Los algoritmos planificadores SCAN y CSCAN son más apropiados para sistemas que pongan una gran carga sobre el disco. Es posible definir un algoritmo óptimo, pero la computación necesaria para un planificador óptimo puede no justificar el ahorro sobre los planificadores SSTF o SCAN. Con cualquier algoritmo planificador, sin embargo, el rendimiento depende mucho del número y tiempo de requerimientos.

En particular, si rara vez la cola tiene más de un requerimiento pendiente, entonces todos los algoritmos planificadores son efectivamente equivalentes. En este caso, el planificador FCFS es también un algoritmo razonable.

La elección de un algoritmo de planificación acorde depende de:

* Cantidad de pedidos
* Como están almacenados los archivos
* Posición de los directorios y sus archivos

**Almacenamiento intermedio de E/S (Buffering)**

Un buffer es un espacio de memoria reservado para almacenamiento temporal de datos, para

acomodar disparidades entre las velocidades de un dispositivo productor de datos y un dispositivo consumidor, o diferencias en tamaños de bloques.

Con las disparidades entre las velocidades existen dos problemas: el primero, es que el programa está bloqueado esperando que se complete la operación de E/S. El segundo problema es que esto interfiere con las decisiones del swapping a través del sistema operativo.

En un buffer se almacenan datos temporalmente mientras se transfieren entre dos dispositivos o entre un dispositivo y una aplicación.

Cuando un proceso realiza un pedido de E/S, es bloqueado hasta que la operación se termine, pero no puede ser intercambiado ya que las posiciones desde o hacia las cuales debe hacerse la transferencia deben continuar en memoria central para ser utilizadas. Para evitar esto a veces es conveniente llevar a cabo las transferencias de entrada por adelantado a las peticiones y realizar las transferencias de salida un tiempo después de la petición. Esta técnica se conoce como almacenamiento intermedio (I/O buffering).

Las razones para el almacenamiento intermedio son:

* Los procesos deben esperar que una E/S termina
* Ciertas páginas deben permanecer en memoria central durante una E/S.

Hay dos tipos de dispositivos de E/S:

**Orientado a bloque**

* Almacenan la información en bloques de tamaño fijo
* Las transferencias son de un bloque a la vez
* Se usa en discos y cintas

**Orientado a flujo**

* Transfieren los datos como flujos de bytes
* Se usa en terminales, impresoras, puertos de comunicación, ratones, y muchos otros dispositivos que no son de almacenamiento secundario.

**Buffer único o sencillo**

Cuando un proceso realiza una petición de E/S, el S.O. le asigna a la operación un buffer en la parte del sistema de la memoria central. Para los dispositivos orientados a bloques, las transferencias de entrada se realizan al buffer del sistema, cuando se ha completado la transferencia el proceso mueve el bloque al espacio del usuario y pide otro bloque. Esta técnica es llamada lectura por adelantado y se realiza con el pensamiento de que el bloque será necesitado más adelante. De esta forma el S.O. será capaz de expulsar al proceso porque la operación de entrada tiene lugar dentro de la memoria del sistema. Para la E/S con dispositivos de flujo el esquema de buffer sencillo puede aplicarse por líneas o por bytes, y se realiza lo mismo que en el de bloques (lectura adelantada).

**Orientado a bloques**

* Las transferencias de entrada se realizan en el buffer
* Se mueve el bloque al espacio del usuario cuando se necesita
* Se mueve otro bloque al buffer: lectura adelantada
* El proceso de usuario puede procesar un bloque de datos mientras se esté leyendo el siguiente bloque de datos
* El intercambio puede ocurrir ya que la entrada está tomando lugar en la memoria del sistema, no en la memoria del usuario
* El S.O. mantiene constancia de las asignaciones de buffers del sistema a los procesos de usuario
* La salida se realiza por el proceso de usuario escribiendo un bloque el buffer y después hacia fuera

**Orientado a Flujo**

* Usa una línea a la vez
* La entrada del usuario de una terminal es una línea a la vez con **cr** (carrier return) indicando el fin de la línea
* La salida a la terminal es una línea a la vez

**Buffer doble**

Como su nombre lo indica en esta técnica se utilizan dos buffers para realizar la función de E/S.

Los procesos transfieren datos desde/hacia un buffer mientras el sistema operativo llena o vacía (según corresponda) el otro buffer. De esta forma el proceso no tendrá que esperar por la E/S, ganando velocidad de procesamiento.

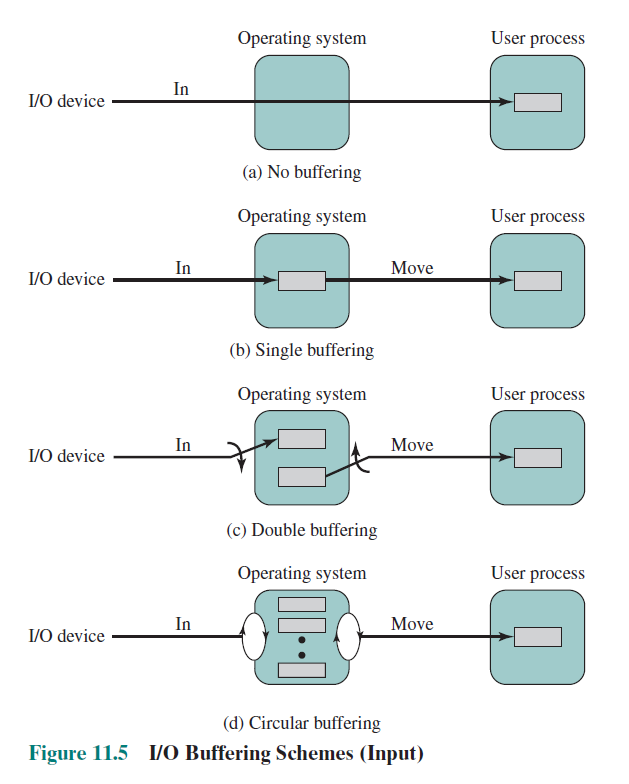
Entonces se asignan a un proceso dos buffers del sistema a la operación de E/S. De esta forma el proceso puede transferir datos hacia o desde un buffer mientras que el S.O. vacía o llena al otro. Esta técnica mejora las transferencias de bloques y de flujos por línea, pero en la operación de flujos por bytes no ofrece ventajas con respecto a un buffer sencillo de doble tamaño.

**Buffer Circular**

Esta técnica se utiliza si el proceso realiza frecuentes ráfagas de E/S, lo que se hace es permitir que el proceso utilice más de dos buffers. Cada buffer individual se considera como una unidad del buffer circular, y son tratados según el modelo de productor/consumidor. Con esta técnica se puede maximizar el desempeño individual del proceso en cuestión.

* Se usan más de dos buffers, cada buffer individual es una unidad en un buffer circular.
* Se basa en el modelo productor consumidor con buffer limitado.

El esquema del buffer doble debería solucionar el flujo de datos entre un dispositivo de E/S y un proceso. Si preocupa el rendimiento de un proceso determinado, sería deseable que las operaciones de E/S fueran capaces de ir al ritmo del proceso. El buffer doble puede ser inapropiado si el proceso lleva a cabo rápidas ráfagas de E/S. En este caso, el problema puede mitigarse usando más de dos buffers. Cuando se emplean más de dos, el conjunto de buffers se conoce con el nombre de *buffer circular*. Cada buffer individual constituye una unidad del buffer circular. Este se basa en el modelo del productor/consumidor con un buffer limitado.



**La Utilidad del Almacenamiento Intermedio**

El almacenamiento intermedio es una técnica que soluciona los problemas de "cuellos de botella" en la demanda de E/S. Sin embargo, no existe un tamaño de los buffers que asegure a un dispositivo de E/S ir al mismo ritmo que un proceso cuando la demanda media del proceso es mayor que la que el dispositivo puede admitir.

Incluso si se dispone de varios buffers, al final todos se llenarán y el proceso tendrá que quedarse esperando tras operar con una determinada cantidad de datos. Sin embargo, en un entorno de multiprogramación, con la variedad de actividades de E/S y de procesamiento que hay que realizar, el almacenamiento intermedio es una herramienta que puede incrementar la eficiencia del sistema operativo y el rendimiento de los procesos individuales.

**Técnicas de E/S**

Hay tres técnicas posibles para las operaciones de E/S. Con la ***E/S Programada*, también llamada de *Polling (Escrutinio)***, los datos son intercambiados entre el procesador y el módulo de E/S. El procesador ejecuta el programa que otorga el control directo de la operación de E/S, incluyendo el sensado del estado del dispositivo, enviar un comando READ o WRITE, y la transferencia de los datos. Cuando el procesador emite un comando al módulo de E/S, debe esperar hasta que la operación de E/S se complete.

Si el procesador es más rápido que el módulo de E/S, se desperdicia tiempo de la CPU. Con **la *E/S dirigida por interrupciones*** (interrupt driven), el procesador emite un comando de E/S, continúa con la ejecución de otras instrucciones, y es interrumpida por el módulo de E/S cuando éste ha completado su trabajo.

Tanto en la E/S programada o por interrupciones, el procesador es responsable de extraer los

datos de la memoria central para el output o almacenar los datos en la memoria central para el input. La alternativa a esto se conoce como ***acceso directo a memoria* (DMA).** En este modo, el módulo de E/S y la memoria central intercambian datos directamente, sin involucrar a la CPU.

**E/S por DMA (ACCESO DIRECTO A MEMORIA)**

**Desventajas de la E/S por Interrupción o Programada**

La E/S por interrupciones, si bien es más eficiente que la E/S programada, todavía requiere de la intervención activa del procesador para transferir los datos entre la memoria y el módulo de E/S, y cualquier transferencia de datos debe recorrer un camino pasando por la CPU. Luego, ambas formas de E/S sufren de dos desventajas inherentes:

* La velocidad de transferencia de E/S está limitada por la velocidad a la cual el procesador puede verificar y servir a un dispositivo.
* El procesador está ocupada administrando la transferencia de E/S; un número de instrucciones debe ser ejecutadas para cada transferencia de E/S

Hay cierto grado de compromiso entre estas dos desventajas. Considere la transferencia de un

bloque de datos. Usando E/S programada, el procesador es dedicado a la tarea de E/S y puede mover los datos a una relativamente alta velocidad, al costo de no hacer nada más. La E/S por interrupción libera al procesador hasta cierto punto a expensas de la velocidad en E/S. No importa cual, ambos métodos tienen un impacto adverso tanto en la actividad del procesador como en la velocidad de transferencia de E/S.

Cuando grandes cantidades de datos deben ser movidos, se requiere una técnica más eficiente: el acceso directo a memoria (DMA).

**Función del DMA**

El DMA requiere un módulo adicional en el bus del sistema. El módulo de DMA es

capaz de simular al procesador y tomar el control del sistema desde la CPU. La técnica opera de la siguiente manera. Cuando el procesador desea leer o escribir un bloque de datos, emite un comando al módulo de DMA, enviándole al módulo de DMA la siguiente información:

* Si la operación solicitada es una lectura o una escritura.
* La dirección del dispositivo de E/S involucrado.
* La posición inicial en memoria a leer o a escribir.
* El número de palabras a ser leído o escrito.

**Transferencia de datos mediante DMA**

Cinco formas de obtener el control del bus:

* **Por ráfagas**: Cuando el DMA toma el control del bus, no lo libera hasta haber transmitido el bloque de datos pedido. Se consigue la mayor velocidad de transferencia, pero se tiene inactiva la CPU (parada del procesador).
* **Por robo de ciclos:** Cuando el DMA toma el control del bus, lo retiene durante un solo ciclo. Transmite una palabra (un byte) y libera el bus. Es la forma más usual. Reduce al máximo la velocidad de transferencia y la interferencia del DMA sobre la actividad del procesador.
* **DMA transparente:** Se elimina completamente la interferencia entre el DMA y la CPU. Sólo se roban ciclos cuando la CPU no está utilizando el bus del sistema. No se obtiene una velocidad de transferencia muy elevada.
* **Por demanda:** El periférico comienza la transferencia por DMA, pero devuelve el control a la CPU cuando no tienen más datos disponibles. Tan pronto como el periférico tiene nuevos datos, vuelve a retomar el control del bus y se continúa así hasta que acaba la transferencia completa del bloque.
* **Dato a dato:** Cada vez que el periférico solicita una transferencia por DMA, se envía un único dato y se devuelve el control a la CPU. El proceso acaba cuando se ha transferido todo el bloque. De utilidad cuando se desea simultanear la ejecución de un programa con la recepción o transmisión de datos a velocidades moderadas. La CPU sigue ejecutando su programa casi con la misma velocidad mientras se efectúa la transferencia de datos.

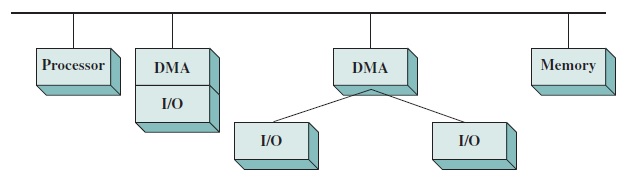
Los ciclos de bus utilizados por el DMA no suponen una interrupción. El procesador no guarda su contexto, puesto que el controlador DMA no altera los registros de la CPU. El procesador para su actividad durante un ciclo de bus.

El procesador luego continúa con otro trabajo. Ha delegado esta operación de E/S al módulo de

DMA, y ese módulo se hará cargo de la operación. El módulo de DMA transfiere el bloque entero de datos, una palabra a la vez, directamente desde o hacia la memoria, sin pasar por el procesador. Cuando la transferencia se ha completado, el módulo de DMA envía una señal de interrupción a la CPU. En consecuencia, el procesador sólo opera al comienzo y al final de la transferencia.

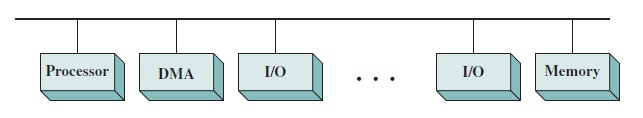
**Modos de conexión del DMA**

**Bus único,**



Todos los módulos comparten el mismo bus del sistema. El módulo de DMA, actuando como una CPU sustituta, utiliza E/S programada para intercambiar datos entre memoria y el módulo de E/S a través del módulo de DMA. Esta configuración, si bien puede ser barata, claramente es ineficiente. Al igual que en la E/S programada controlada por la CPU, cada transferencia de una palabra consume 2 ciclos de bus.

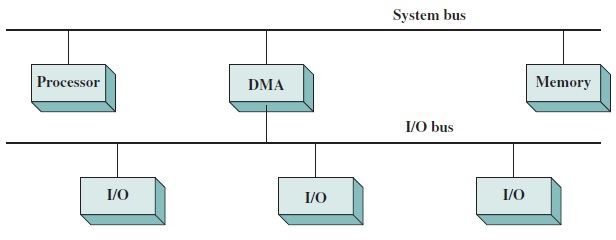
**DMA integrado simplemente a un bus**



El número de ciclos de bus requeridos puede ser substancialmente bajado integrando las funciones de DMA junto a las funciones de E/S. Como se indica en la imagen, esto significa que hay una trayectoria entre el módulo de DMA y uno o más módulos de E/S que no incluye al bus del sistema.

La lógica del DMA puede también ser parte de un módulo de E/S, o puede ser un módulo separado que controla uno o más módulos de E/S.

**DMA conectado a un bus de E/S y al bus del sistema**



Se puede avanzar un paso más en este concepto conectando los módulos de E/S al módulo de DMA a un bus de E/S. Esto reduce el número de interfaces a uno en el módulo de DMA y provee una configuración fácilmente expansible.

En este caso y en el anterior (DMA integrado al bus del sistema), el bus del sistema que el módulo de DMA comparte con el procesador y la memoria es utilizado por el DMA sólo para intercambiar datos con la memoria. El intercambio de datos con los módulos de E/S toma lugar fuera del bus del sistema.

**Sistemas de archivos (File Systems)**

**def. Archivo:**  unidad lógica de almacenamiento. Es una abstracción de las características físicas de los dispositivos del almacenamiento, hecha por el sistema operativo.

**def. File System:** es el modulo encargado de administrar el espacio de almacenamiento en un dispositivo y las estructuras abstractas de datos.

El sistema de archivos es la única forma en que un usuario o aplicación puede acceder a archivos. Es deber del mismo ocultar la complejidad de la manipulación de la estructura abstracta de datos, y su almacenamiento, y proporcionarle una visión simplificada y uniforme de un nombre desde el punto de vista lógico.

**def. Sector o bloque físico:** hace referencia a la subdivisión de una pista de un disco duro.

**def. Sector o bloque lógico:** es una subdivisión lógica realizada por el sistema operativo. Un sector lógico puede ser igual a uno físico (relación 1 a 1), o puede ser un conjunto de sectores físicos (lo cual recibe el nombre de clúster).

**Objetivos de un FS**

* Satisfacer los requerimientos y las necesidades del usuario en cuanto a administración de datos, que incluye: a) el almacenamiento de los datos y b) poder efectuar las operaciones anteriormente mencionadas (operar con archivos, registros y campos).
* Garantizar (en la medida de lo posible) la validez de los datos almacenados en cada archivo.
* Optimizar el desempeño: a) desde el punto de vista del sistema: overall throughput (rendimiento de todos los componentes) y b) desde el punto de vista del usuario: en términos de tiempos de respuesta cortos. Como se ve, son objetivos contrapuestos.
* Proveer soporte de E/S para una amplia variedad de dispositivos de almacenamiento.
* Minimizar (o eliminar) la posibilidad de pérdidas o destrucción de datos.
* Proveer un conjunto standard de rutinas de interfaces de E/S.
* Proveer soporte de E/S para usuarios múltiples, en el caso de sistemas multiusuarios.

En cuanto al primer objetivo los requerimientos dependerán de la variedad de aplicaciones o ámbitos en el cual se use el sistema de computación.

**Requerimientos mínimos de un FS para un sistema de propósito general**

Cada usuario debe poder:

* Crear, borrar, leer y modificar archivos.
* Tener acceso controlado a archivos de otros usuarios.
* Controlar que tipo de acceso permite sobre sus archivos.
* Reestructurar sus archivos en forma apropiada a cada problema.
* Mover datos entre archivos.
* Efectuar copias de resguardo y restaurar sus archivos en caso de daño.
* Acceder a sus archivos por medio de NOMBRES SIMBÓLICOS.

**Problemáticas que debe resolver un FS**

Estos problemas surgen cuando se desea administrar la memoria secundaria:

* Determinación de espacio libre, ocupado y defectuoso.
* Determinación del espacio ocupado por un archivo.
* Determinación de la dirección en el soporte de un archivo.
* Organización del directorio y de los archivos.
* Los Métodos de Accesos.
* Protección.

**Alternativas para la asignación de espacio a los archivos (File Allocation)**

Hay algunos aspectos a tener en cuenta al momento de asignarle espacio a un archivo:

* **Preasignación vs. Asignación Dinámica:** en la preasignación se debe conocer de antemano la longitud máxima a ocupar del archivo. En algunos casos no es difícil saberlo, pero en muchos otros sí. Se hace una estimación grosso modo del posible tamaño máximo, otorgándole al archivo una medida mayor (con lo cual se desperdicia espacio) para que los procesos nunca se aborten o suspendan por falta de espacio.   
    
  En cambio, la asignación dinámica tiene la ventaja de que el espacio se va otorgando a medida que el archivo va creciendo.
* **Longitud del tamaño del bloque:** puede ser un solo bloque para todo el archivo, o asignársele uno cada vez que el archivo lo pida. Hay un compromiso de eficiencia:   
    
  1) El espacio contiguo aumenta bastante el rendimiento, más si el archivo es accedido de forma total y secuencial.  
    
  2) Asignar muchos bloques chicos aumentaría notablemente la longitud de la File Allocation Table, como también la longitud de la tabla de espacios libres.  
    
  3) Trabajar con bloques lógicos de longitud fija (por ejemplo, de N sectores físicos) simplifica la reasignación de espacios libres.   
    
  4) Operar con bloques lógicos de longitud variable minimiza la fragmentación interna.   
    
  Estos cuatro ítems nos llevan a dos alternativas:

**Bloques largos y contiguos, de longitud variable:** La longitud variable evita derroches, la contigüidad hace que las FAT sean pequeñas. Como desventaja, el espacio es muy difícil de reusar (reasignarse).

**Bloques pequeños de longitud fija proveen de gran flexibilidad:** Su asignación requiere de tablas grandes o estructuras complejas. Se abandona la contigüidad, los bloques se asignan cuando se necesitan.

**Catalogación de la información**

Para poder asignar y desasignar el sistema operativo divide al disco en tres áreas:

* **Área de datos fijos (Fixed Data Area):** esta área contiene datos tales como el comienzo y fin de cada partición, tamaño, estado de la partición (activa o inactiva), etc., Además contiene el sector de booteo, la cantidad de bloques que dispone el disco, cantidad de entradas en el directorio, cantidad de sectores por bloque, dirección de comienzo del directorio, etc.  
    
  Es inicializada durante el formateo de alto nivel. Son datos críticos que deben ser apropiadamente protegidos por el sistema operativo.
* **Área de catálogo:** es el área donde el sistema operativo almacena el directorio, la tabla de asignación de espacio, la lista de espacio libre del disco y la lista de sectores defectuosos (inutilizados).   
    
  Una vez que se crea esta área, su tamaño no se modifica, ya que modificar el tamaño del área de catálogo supondría mover todos los datos del área que le sigue.
* **Área de datos:** finalizada el área de catálogo, el S.O asigna todo el resto del disco libre para el almacenamiento de los archivos del usuario.

**Administración del espacio libre**

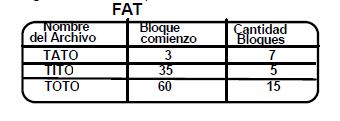
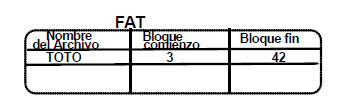
**def. File Allocation Table:** la FAT es una tabla dentro del área de catálogo que se encarga de indicar cuáles son los sectores que un archivo ocupa. Mantiene constancia de las secciones asignadas a un archivo.

Se necesita saber qué espacios y sectores de un disco están libres para ser asignados. De otra forma, no podríamos guardar en disco ningún archivo porque no sabríamos qué lugares están disponibles para hacerlo.

Para eso existen las siguientes formas:

* **Bit Map o Bit Vector:** se implementa como un vector de bits en la FAT. Cada bloque del disco se representa con un bit. Si el bloque está libre, se le asigna un bit 0, caso contrario, el bit valdrá 1.   
    
  Este es el sistema que generalmente se usa, ya que no posee grandes desventajas. Es sencillo de implementar, es fácil de encontrar un bloque (o un grupo contiguo de bloques) libres, y el tamaño que ocupa es muy reducido (1 bit por cada bloque del disco).
* **Lista enlazada de bloques libres:** se implementa una lista enlazada en la que cada puntero apunta al siguiente bloque libre. Dependiendo del método de asignación que se use, se tendrán que ajustar los punteros de la lista, y esto es algo costoso.   
    
  Sus ventajas son que ocupa un espacio mínimo. Simplemente se almacena el puntero al comienzo de la lista.  
    
  Sus desventajas es que resulta poco eficiente, ya que para recorrer toda la lista se debe leer cada bloque del disco, lo que implica un gran tiempo de E/S. Además, es altamente vulnerable, ya que, si se corta un enlace de la lista, el sistema pierde totalmente el control sobre el espacio libre.
* **Bloque de direcciones libres:** en uno o más bloques lógicos reservados del disco, se mantiene una lista de todos los aquellos bloques libres. No ocupa tanto espacio en disco, es menor que el 1% del espacio total. Si bien es muy difícil llevar a memoria central esta lista, si se la trata como un stack o una cola FIFO puede evadirse el hecho de tener que cargarla completamente.   
    
  Las ventajas de este método es que, con solo leer ese bloque lógico, podemos tener conocimiento de todos aquellos bloques libres del disco. Además, mientras más ocupado esté el disco, menos espacio requerirá la lista.  
    
  La desventaja es que cuando disco está libre, la lista es muy grande y se tarda mucho en recorrerla.
* **Bloques de direcciones libres contiguas:** es una modificación del método anterior, en vez de guardarse bloques que tengan todas las direcciones libres, se guardan bloques que tengan N direcciones libres contiguas.   
    
  La ventaja es que ocupa menos almacenamiento que el método anterior, porque habrá menos entradas en la lista.  
    
  La desventaja es que solamente es aplicable para almacenamiento contiguo.

**Métodos de asignación de espacio para los archivos**

* **Asignación contigua fija:** es el método de preasignación que se habló anteriormente. Requiere que cada archivo ocupe un conjunto de direcciones contiguas o consecutivas.   
    
  La tabla FAT sería algo así:  
    
    
    
    
    
    
    
    
    
    
  **Ventajas:** es muy fácil acceder a los archivos, solo hay que leer la zona de catálogo una vez para saber dónde comienza y dónde termina. Es ideal para accesos secuenciales ya que los archivos están guardados de forma contigua.   
  **Desventajas:** a medida que el disco se llena, es más difícil encontrar espacio para un nuevo archivo. Tendríamos que buscar N bloques consecutivos en la FAT y eso tarda tiempo.   
    
  Cuando un archivo se crea, no se sabe cuántos bloques va a ocupar. Además, si el archivo tiene que crecer y no tiene más espacio en su bloque final, habría que relocalizarlo.   
    
  Es común que haya fragmentación externa (puede haber espacio suficiente para un archivo, pero que este no sea consecutivo).
* **Asignación contigua dinámica:** la asignación dinámica trata de solucionar el problema de cómo satisfacer un pedido de tamaño de N bloques libres. Aplica los algoritmos de First-Fist, Best-Fit, y Worst-Fit. Estos algoritmos sufren de fragmentación externa.   
    
  Tanto la asignación dinámica como la preasignada pueden llevar a casos donde sea necesario compactar (desfragmentar) la memoria.
* **Asignación enlazada:** cada archivo es una lista enlazada de bloques en disco. Los bloques pueden estar distribuidos en cualquier parte, no necesariamente contiguamente.  **Ventajas:** no posee fragmentación externa, cualquier bloque libre poder usado para satisfacer un pedido.   
    
  Crear un archivo usando esta técnica es fácil ya que solo es necesario crear una entrada en la FAT. No es necesario declarar la longitud al crearlo, porque puede crecer en tanto existan bloques libres. Esto elimina la compactación. **Desventaja:** solo es efectivo para accesos secuenciales, ya que para acceder a un al iesimo bloque, debemos recorrer todos los anteriores.   
    
  Los punteros ocupan un espacio extra, por lo que los archivos requerirán más espacio para ser guardados.  
    
  Altamente vulnerable, ya que si un puntero se pierde, la lista enlazada se rompe, por lo que se llevaría a una pérdida de información.
* **Asignación indexada:** cada archivo tiene su propio bloque de índices, que es un vector de direcciones de bloques en el disco. La tabla FAT contiene la dirección del bloque de índices.   
  El bloque de índices puede recorrerse secuencialmente, o también de forma directa yendo al índice que queramos.   
    
  Para archivos grandes, podemos usar varios bloques de índices. O también, podemos usar indirecciones, y tener bloques de índices que apuntan a otros bloques de índices, que a su vez estos apuntan a los bloques del archivo.   
    
  En el caso de un archivo pequeño, que quepa en un sólo bloque lógico, esto representa un desperdicio de 100% de espacio (un bloque lógico para el i-nodo y otro para los datos); para archivos más grandes, la sobrecarga relativa disminuye, pero se mantiene siempre superior a la de la asignación ligada.  
  ****  
  **Ventajas:** soporta acceso directo y acceso secuencial.   
    
  No posee fragmentación externa, cualquier bloque de disco puede ser utilizado para almacenamiento o para índice.  
    
  No es necesario declarar el tamaño del archivo al momento de crearlo.  
    
  **Desventajas**: el espacio que ocupan los bloques de índices es mayor que el utilizado por la asignación enlazada. Un archivo que ocupa dos bloques desperdicia un bloque completo para punteros, con asignación enlazada el espacio perdido sería de sólo 4 bytes.  
    
  Para archivos muy grandes, acceder a un bloque puede involucrar muchas indirecciones, lo que es una pérdida de tiempo. Por estas razones este esquema es ineficiente para bases de datos grandes.

**def. I-nodo:** information node (nodo de información); Es la estructura en disco que guarda los metadatos de cada uno de los archivos, proporcionando un vínculo entre la entrada en el directorio y los datos que lo conforman.

La información almacenada incluye todos los metadatos relacionados con el archivo a excepción del nombre (mismo que radica únicamente en el directorio): los permisos y propietarios del archivo, sus fechas de creación, última modificación y último acceso, y la relación de bloques que ocupa en el disco.

Esta separación entre directorio e i-nodo permite a un mismo archivo formar

parte de distintos directorios.

**def. Extent:** un extent es un espacio contiguo de sectores lógicos dentro del disco. Es un rango de sectores. Generalmente se utilizan para acortar hacer más cortas las listas de punteros de los i-nodos. En vez de ir nombrando puntero por puntero, nombramos un extent, que es un rango de punteros contiguos…

Los extents son usados para estar al tanto de dónde están localizados los contenidos de un archivo. Idealmente, el contenido completo de un archivo debería ser guardado en un espacio contiguo, de forma que podría ser descripto por un solo extent. Pero si el archivo está fragmentado, cada porción del mismo es descripta en extents separados.

**Estructuras de Directorios**

**Estructura de datos utilizadas para los directorios**

Se puede utilizar una lista lineal que refleje las entradas de un directorio. Requiere de una búsqueda lineal para encontrar una entrada en particular. Esto es simple de programar, pero la ejecución demanda mucho tiempo. Crear un archivo implica recorrer toda la lista para asegurar que no esté duplicado… lo mismo para encontrar un archivo y borrarlo.

Puede mejorarse utilizando un ordenamiento clasificado (árbol binario), y así implementarse una búsqueda binaria… pero esto implica programar algoritmos más complejos. Además siempre se debería mantener la lista ordenada, y eso es costoso.  
  
También podrían utilizarse una tabla hash, aunque debe tenerse cuidado en el caso de colisiones - situaciones donde dos nombres de archivos pretenden apuntar a una misma dirección.

**Estructura de nivel único (single level directory)**

Solo se posee un directorio raíz donde todos los archivos son almacenados. Es una estructura muy limitada, ya que no puede haber nombre repetidos, y si la cantidad de archivos crece, esto se vuelve complicado de evitar.  
  
**Ventajas:**

* Fácil implementación.
* Fácil de usar por el usuario.

**Desventajas:**

* Cuando crece el número de archivos o el número de usuarios la regla de unicidad de

nombres simbólicos es cada vez más difícil de cumplir, y la búsqueda dentro del

directorio se torna más lenta.

* No permite agrupar archivos con características comunes. Por ejemplo, sería deseable

separar los archivos de datos de los archivos del sistema.

* La mayor desventaja de esta estructura es la confusión de los nombres de archivo

entre diferentes usuarios. La solución standard es crear directorios separados para

cada usuario. Especialmente en grandes sistemas, este directorio de usuarios es una organización lógica antes que física, ya que todos los archivos siguen estando físicamente en el mismo dispositivo.

**Estructura de dos niveles**



Existe un directorio raíz, y a su vez, cada usuario tiene su propio directorio. Esta estructura puede verse como un árbol de dos niveles.

En sí, cada User File Directory (directorio de archivos del usuario) se maneja de igual forma que un directorio de nivel único (ya sea con una lista lineal, binaria, o hash).

Existen comandos y programas especiales que crean y borran los UFD. Al ser ejecutado, se crea una entrada en el Master File Directory (archivo que contiene todos los directorios).   
  
Un caso especial sucede con los programas del sistema (utilitarios, compiladores, subrutinas,

cargadores, etc.). Para no duplicar todos estos programas en cada directorio de usuario para su uso, se define un usuario especial, por ejemplo, USUARIO0 donde se cargan todos los programas del sistema. Esto complica un poco las búsquedas, pero soluciona el problema de espacio en disco. Al tratar un usuario ejecutar un programa, el SO busca primero en el directorio del usuario que invocó el programa y si no lo encuentra lo busca en ese directorio definido especialmente.

**Ventajas:**

* Permite que varios usuarios trabajen con sus propios archivos sin interferir con los

demás. De esta forma diferentes usuarios pueden tener archivos con el mismo nombre.

* Para crear o borrar archivos, el sistema operativo limita la búsqueda al UFD, de esta forma no se puede borrar accidentalmente un archivo de otro usuario.

**Desventajas:**

* Así como no deja que dos usuarios se interfieran, tampoco los deja trabajar en forma conjunta. Si dos usuarios trabajan en el mismo proyecto, deberían poder compartir la información contenida en sus archivos.
* No permite agrupar archivos. Un usuario puede querer agrupar sus archivos, y no

tenerlos todos juntos en una sola lista.

**Estructura de árbol**



Una vez comprendida la estructura de dos niveles como un árbol de dos niveles, la generalización natural es extender esta estructura a un árbol arbitrario de n niveles.

Este esquema permitirá a los usuarios crear sus propios subdirectorios y organizar los archivos

como prefieran.

Los nombres de camino (ruta, path) pueden ser de dos tipos:

* **Completos:** Comienza en la raíz y sigue un camino hasta el archivo especificado.
* **Relativos:** Define un camino desde el directorio actual.

La habilidad de que el usuario defina su propia estructura de subdirectorios les permite a ellos imponer su propia estructura de archivos. Como resultado se puede tener directorios separados para archivos que están asociados por diferentes tópicos o distintos tipos de información (por ejemplo, el directorio programas puede contener todos los archivos fuente, etc.)

Un punto interesante es este método es como se va a manejar el borrado de un directorio. Si

esta vacío, simplemente se puede borrar. Sin embargo, si no está vacío, puede contener archivos e incluso subdirectorios. Para el borrado existen dos soluciones. Algunos sistemas no borraran un directorio si no está vacío. Esta solución puede conllevar una cantidad substancial de trabajo de borrando archivos. Una alternativa es suponer que, ante un pedido de borrado de un directorio, debe borrarse archivos y subdirectorios contenidos.

**Ventajas:**

* Todas las características mencionadas acerca de esta estructura constituyen ventajas sobre las estructuras anteriores.

**Desventajas:**

* No permite que los archivos o directorios sean compartidos por varios usuarios. Permite la duplicación de archivos que no es lo mismo. En el siguiente punto se tratará este tema.

**Estructura de grafos acíclicos**



¿Qué pasa si queremos compartir el mismo archivo o directorio **sin** duplicarlo? Suponete que dos programadores están trabajando en un mismo proyecto y necesitan trabajar con los mismos archivos de un directorio. Este directorio en común debería ser compartido.

Los archivos o directorios compartidos estarán repetidos dos o más veces dentro del file system. Pero en vez de ser copias del archivo, son más bien referencias al mismo archivo.

Con una estructura de árbol no se pueden compartir de esta forma los archivos o directorios. En cambio, un gráfico acíclico permite que los directorios tengan subdirectorios y archivos compartidos.

**Ventajas:**

* La estructura de gráficos acíclicos es mucho más flexible que la estructura de árbol.
* Permite compartir archivos y/o subdirectorios.

**Desventajas:**

* El sistema operativo debe asegurarse que no existan ciclos en los vínculos.
* Es compleja la implementación.
* Distintos nombres de archivo pueden referirse al mismo archivo, con lo cual cualquier actividad contable, backup, búsqueda de archivos, etc. se complica ya que hay que vigilar de no contabilizar dos veces el mismo archivo.
* Cuando se borra un archivo compartido, la actualización de los vínculos es compleja.

La creación de archivos y subdirectorios compartidos puede implementarse de varias maneras:

* **Hard-Linking:** presente en la mayoría de los sistemas operativos modernos. Se crea una nueva entrada dentro de un directorio que se la conoce como link. Este link es un puntero a otro archivo a subdirectorio. De esta forma, un archivo puede ser invocado con diferentes nombres (alias).   
    
  Los sistemas operativos deciden como implementar el borrado de un archivo que está hard-linkeado. El borrado de un link no necesariamente implica borrar el archivo original. Pero si borramos el archivo original, varias referencias podrían quedar muertas. Estas podrían almacenarse hasta que alguien intente usarlas, y ahí eliminarlas.

La mayoría de los sistemas operativos no permiten hacer hard-linking a directorios (carpetas), ya que eso podría crear ciclos.

* **Soft o symbolic linking:** en vez de crearse un puntero al archivo, se crea un archivo que sirve de acceso directo al mismo. En sí, los links simbólicos contienen la ruta relativa o completa hacia el archivo destino. El sistema operativo se encarga de procesar la ruta y abrir el archivo deseado.   
    
  A diferencia de los hard-links, los soft-links pueden “romperse” y hacer referencia a un archivo que ya no existe.
* **Duplicar las entradas:** se crean copias idénticas del archivo en directorios diferentes. Es hacer un copy y paste. Los cambios que se hagan en una de las copias no se ven reflejadas en la otra. No sirve de mucho.

**Directorio de grafos generalizado**



No hay una organización explícita, pueden aparecer ciclos.

Necesitan una construcción especial, necesitan un elemento que atraviese todo el sistema de archivos para evitar el problema de las autoreferencias (cuando un el número de referencias es distinto de cero y físicamente no es posible acceder al archivo)

Para solucionar esto se examina todo el sistema de archivos, aquel archivo al que se pueda acceder queda marcado y luego se borran los que no estén marcados. Esto se conoce como Garbage collection. Es un sistema lento y pesado.

**Subdirectorios**

Los subdirectorios son en realidad archivos de un tipo especial: una suerte de archivos estructurados gestionados por el sistema operativo. Dependiendo del sistema de archivos que se use, los directorios pueden ser o bien un archivo que tenga entrada en la FAT con algo que los distinga (un bit, por ejemplo), o bien ser un archivo representado a través de un I-Nodo.

Un directorio es almacenado en disco exactamente como cualquier otro archivo. Y dado que es tratado tal cual si fuera un archivo normal, estará sujeto también a la fragmentación: conforme se agreguen entradas al directorio, éste crecerá. Llegado el momento, requerirá bloques lógicos adicionales. Y si un directorio termina disperso por todo el disco, resultará –como cualquier otro archivo– más lento leerlo y trabajar con él. Siempre que se abra un archivo dentro de un directorio grande, o que se le recorra para abrir algún archivo en un subdirectorio suyo, el sistema tendrá que buscar todos sus fragmentos a lo largo del disco.

Otros sistemas de archivos más modernos han resuelto este problema mediante los *grupos de asignación*: los directorios del sistema son esparcidos a lo largo del volumen, y se intenta ubicar los archivos cerca de los directorios desde donde son referidos.

Esto tiene como consecuencia que los archivos que presentan cercanía temática (que pertenecen al mismo usuario o proyecto, o que por alguna razón están en la misma parte de la jerarquía del sistema) quedan ubicados en disco cerca unos de otros (y cerca de sus directorios). Y dado que es probable que sean empleados aproximadamente al mismo tiempo, esto reduce las distancias que recorrerán las cabezas.

Además, al esparcir los archivos, se distribuye también mejor el espacio libre, por lo cual el efecto de los cambios de tamaño de un archivo en lo relativo a la fragmentación se limita a los que forman parte del mismo bloque de asignación.

Los sistemas de archivos que están estructurados siguiendo esta lógica de grupos de asignación no evitan la fragmentación, pero sí la mayor parte de sus consecuencias negativas.

**Borrado de un archivo**

Cuando se le solicita a un sistema de archivos que implementa FAT eliminar un archivo, éste no se borra del directorio, ni su información se libera de la tabla de asignación de archivos, sino que se marca para ser ignorado. Ni la entrada de directorio, ni la cadena de bloques lógicos correspondiente en las tablas de asignación, son eliminadas —sólo son marcadas como disponibles. El espacio de almacenamiento que el archivo eliminado ocupa debe, entonces, ser sumado al espacio libre que tiene el volumen/disco.

Es sólo cuando se crea un nuevo archivo empleando esa misma entrada, o cuando otro archivo ocupa el espacio físico que ocupaba el que fue eliminado, que el sistema operativo marca realmente como desocupados los bloques lógicos en la tabla de asignación.

Este es un ejemplo de un mecanismo flojo (en contraposición de los mecanismos ansiosos, como los vistos en la sección). Eliminar un archivo requiere de un trabajo mínimo, mismo que es diferido al momento de reutilización de la entrada de directorio.

**La tabla de asignación de archivos (FAT)**

**def. Cluster:** un clúster es un grupo de sectores físicos contiguos. En sí, un clúster es un bloque lógico.

La familia FAT obtiene su nombre de la tabla de asignación de archivos (file allocation table), que aparece antes del directorio, en los primeros sectores del disco. Cada byte de la FAT representa un cluster en el área de datos; cada entrada en el directorio indica, en su campo correspondiente, cuál es el primer cluster del archivo.

Ahora bien, conforme se usa un disco, y los archivos crecen y se eliminan, y van llenando los espacios vacíos que van dejando, FAT va asignando espacio conforme encuentra nuevos clusters libres, sin cuidar que sea espacio continuo. Los apuntadores al siguiente cluster se van marcando en la tabla, cluster por cluster, y el último de cada archivo recibe un valor especial (End Of File).

Dentro de la entrada de directorio de cada archivo está prácticamente su i-nodo completo: la información de permisos, atributos, fechas de creación, y muy particularmente, el apuntador al cluster de inicio. Esto resulta en una de las grandes debilidades de FAT: la tendencia a la fragmentación.

**Virtualización**

**def. Virtualización:** virtualizar es simular un recurso que no tenemos a través de software. Más específicamente, es presentar a un sistema elementos que se comporten de la misma forma que un componente físico (hardware), sin que exista en realidad —un acto de ilusionismo, en el cual se busca presentar el elemento de forma tan convincente que la ilusión se mantenga tanto como sea posible.

**def. Anfitrión o Host:** El hardware o sistema real, que ofrece el mecanismo de virtualización.

**def. Huésped o Guest:** El sistema o las aplicaciones que se ejecutan en el entorno virtualizado.

La virtualización surge debido al sprawl de servidores. Es decir, en la década de los 80 y 90, cada vez que necesitabas correr una aplicación (un sistema de correos electrónicos, una página web, un adm. de usuarios) comprabas todo el hardware necesario y configurabas un servidor a parte solo para correr esa aplicación. Esto resultó ser muy costoso y un desperdicio de recursos.

Lo que busca solucionar la virtualización es, con un mismo conjunto de recursos, abrir la posibilidad de darles un mejor uso, un mejor provecho. Así, implementando virtualización, podemos tener varias aplicaciones distintas ejecutándose al mismo tiempo. Sí, con un poco de penalidad, pero muchísimo más barato que teniendo que comprar un servidor nuevo.

**Emulación**

Emular consiste en implementar en software algo que se haga pasar como el hardware de un sistema específico (típicamente diferente a la del anfitrión). La emulación no solo consiste en traducir el lenguaje y la estructura de un procesador a otro, sino que también se emulan todos los periféricos, buses, discos, y puertos del sistema que se quiere emular.  
  
Pensá en cualquier emulador de consolas. Se emula todo el hardware de la consola, no solo el procesador.

Hoy en día, la emulación se emplea para hacer desarrollos cruzados, más que para ejecutar software ya escrito y compilado. Esto es debido a que la mayoría de los dispositivos que se venden en la actualidad son sistemas embebidos o dispositivos móviles. Es imposible desarrollar el sistema operativo de un celular… dentro de ese celular. Por lo que se utiliza la emulación para emular la arquitectura de hardware del celular, mientras que se desarrolla en un sistema más potente y robusto.

**Emulación de arquitecturas inexistentes**

Una nota de color sobre la emulación es que se empezaron a emular arquitecturas de hardware que nunca fueron implementadas en la vida real.   
  
Un ejemplo de esto es lo que hizo Java con su Java Virtual Machine. La JVM es una arquitectura de hardware emulada. Cuando escribimos un programa en Java, al compilarlo, lo que hacemos es producir un lenguaje de bajo nivel llamado bytecode. Este bytecode es interpretado por la JVM que, a su vez, lo traduce a lenguaje de ensamblador para que sea ejecutado por el procesador.

Lo importante de esto es que la JVM es una arquitectura de hardware ficticia completamente emulada, que corre encima de un hardware físico real. Es una capa de abstracción. La ventaja de esto es que un programa de Java va a poder ser ejecutado en cualquier arquitectura de hardware que tenga un JVM programada.

**¿Emulación o simulación?**

Emular significa imitar las acciones de otro, procurando igualarlas e incluso excederlas. Esto significa que un emulador reproduce todos los procesos internos que realizaría el sistema

nativo, y busca cubrir todos los comportamientos respectivos implementando los mismos mecanismos.

Simular, por otra parte, significa representar algo, fingiendo o imitando lo que no es. Un sistema simulador simula o finge las áreas de determinado sistema que interesan al usuario; puede emplear datos precargados para generar ciertas respuestas, obviando los procesos que los

generarían.

A diferencia de los ejemplos presentados a lo largo de esta sección, que llevan a ejecutar software arbitrario para la plataforma destino buscando idealmente que éstos no detecten siquiera una diferencia en comportamiento, un simulador puede presentar mucho mayor detalle en determinadas áreas, pero no realiza las funciones sustantivas del sistema simulado. Por ejemplo, es muy común (incluso para el entrenamiento de pilotos reales) el uso de simuladores

de vuelo; estos programas pueden representar una cabina equivalente a la de un avión real, con todos sus monitores y controles, pero nadie esperaría que lo trasladen de un lugar a otro.

**Emulación vs Virtualización**

En la emulación, el sistema operativo Guest NUNCA es ejecutado en el procesador/hardware. Todas las instrucciones de un emulador son enviadas al sistema operativo Host, y este es el que las envía al CPU para su ejecución.

En la virtualización, el sistema operativo Guest *usa* el procesador sin pasar por el sistema Host, salvo que con unos límites. El CPU ejecuta las instrucciones en un modo “virtualizado”. Cada vez que el sistema Guest necesite algún recurso, el CPU llama al S.O Host (syscall) y es este el que procesa la solicitud.

**Virtualización asistida por hardware**

**El hipervisor**Partiendo del concepto de que el sistema operativo es el *supervisor* de la actividad de los usuarios, yendo un paso más hacia arriba, se fue popularizando el nombre de *hipervisor* para el programa que administra y virtualiza a los supervisores. Algunas características primarias que definen qué es un hipervisor son:

* Es únicamente un micro-sistema operativo, dado que no cubre muchas de las áreas clásicas ni presenta las interfaces abstractas al usuario final —sistemas de archivos, mecanismos de comunicación entre procesos, gestión de memoria virtual, evasión de bloqueos, etcétera.
* Se limita a gestionar bloques de memoria física contiguos y fijos, asignación de dispositivos y poco más que eso.
* Normalmente no tiene una interfaz usuario directa, sino que es administrado por medio de llamadas privilegiadas desde alguno de los sistemas operativos huésped.

**Virtualización en x86**

**def. x86:** es como se le conoce comúnmente a la familia de procesadores compatibles con las instrucciones Intel 8086. Engloba a los procesadores de 32 bits que dominaron el mercado durante los 90 y parte de la década del 2000.

Si bien la virtualización existía desde hace tiempo, era para computadoras y arquitecturas específicas, y era algo muy caro y costoso de implementar.

En 2005, Intel lanzó procesadores con las extensiones necesarias para la virtualización. Esta tecnología recibió el nombre de Vanderpool Technology (o Intel VT/X). AMD hizo lo mismo al año siguiente, bajo el nombre de AMD-V.  
  
Estas tecnologías permiten que una computadora de escritorio como la que tenés en tu casa tengan el soporte necesario para llevar a cabo la virtualización asistida por hardware.

En la vida real usamos varios dispositivos periféricos para realizar nuestras tareas en la computadora. La virtualización nos permite emular ciertos dispositivos básicos, sin embargo, dispositivos más complejos necesitan el uso de algún controlador.

Y este último punto permite un acercamiento mayor a una de las ventajas que ofrecen los sistemas operativos virtualizados: la estabilidad en cuanto a los controladores.

Al emplear el sistema operativo huésped únicamente controladores (drivers) ampliamente probados y estabilizados a lo largo de muchos años, la estabilidad que ofrece una máquina virtualizada supera muchas veces a la que obtendría ejecutándose de forma nativa. Claro, el conjunto de máquinas virtuales que se ejecute dentro de un sistema anfitrión sigue siendo susceptible a cualquier inestabilidad del mismo sistema anfitrión, sin embargo, es mucho menos probable que un programa mal diseñado logre congelarse esperando respuesta del hardware (emulado), y mucho menos afectar a los demás huéspedes.

**Hipervisor Bare Metal (al desnudo)**

Como se definió anteriormente, un hipervisor es un tipo de software de virtualización que soporta la creación y la gestión de máquinas virtuales. Los hipervisores traducen las peticiones entre los recursos físicos y los recursos virtuales, haciendo así posible la virtualización.

Cuando un hipervisor es instalado directamente en el hardware de la máquina física, como puente entre el hardware y el sistema operativo, es llamado un hipervisor bare metal. Los hipervisores de este tipo tiene acceso directo al hardware, lo que resulta en mejor rendimiento, escalabilidad, y estabilidad.   
  
Una desventaja de este tipo de hipervisores es que el soporte de hardware es limitado. Los hipervisores bare metal necesitan drivers para cada dispositivo, pero generalmente vienen con pocos drivers implementados.

Son usados mayormente para centros de datos (data centers) de empresas, porque usualmente vienen equipados con características avanzadas de gestión de recursos, alta disponibilidad, y seguridad.

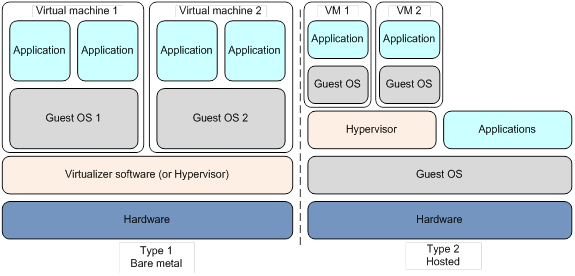
**Hypervisor Host Based**

A diferencia de los hipervisores bare metal, los host based requieren que primero instalemos un sistema operativo.

Este tipo de hipervisores son básicamente aplicaciones que se instalan en un host anfitrión y que permiten la ejecución de un S.O invitado. Esta forma de virtualización provee mejor compatibilidad de hardware, ya que es el S.O host el encargado de manejar los drivers antes que el hipervisor.

Sin embargo, existe la desventaja de no tener acceso directo al hardware. Para acceder al hardware se debe pasar por el S.O anfitrión, lo que produce overhead de recursos y puede degradar el rendimiento de la máquina virtual. También, debido a que hay otros procesos y servicios ejecutándose en el S.O anfitrión, se produce una competencia por los recursos (¿a quién se le dan los recursos, al hipervisor, o a los procesos del host?).

Este tipo de hipervisores son generalmente usados por gente como VOS, o en entornos donde realmente no se necesita mucho rendimiento y solo se busca probar algo. Es más que nada la virtualización que usas en computadoras de escritorio.



**Paravirtualización**

La virtualización asistida por hardware, por conveniente que resulte, sigue presentando algunas desventajas:

* No todos los procesadores cuentan con las extensiones de virtualización. Si bien cada vez es más común encontrarlas, es aún en líneas generales un factor de diferenciación entre las líneas económicas y de lujo.
* La capa de emulación, si bien es delgada, conlleva un cierto peso.
* Si bien es posible virtualizar arquitecturas como la x86, hay muchas otras para las cuales no se tienen las extensiones hardware necesarias.

La paravirtualización, o virtualización asistida por el sistema operativo, parte de un planteamiento distinto: en vez de engañar al sistema operativo para que funcione sobre un sistema que parece real pero no lo es, la paravirtualización busca hacerlo con pleno conocimiento y cooperación por parte de los sistemas huéspedes.

Esto es, la paravirtualización consiste en alojar sistemas operativos huésped que, a sabiendas de que están ejecutando en hardware virtualizado, **no hacen llamadas directas a hardware, sino que las traducen a llamadas al sistema operativo anfitrión.**

Los sistemas operativos huésped bajo un entorno paravirtualizado saben que no están ejecutando sobre hardware real, por lo que en vez de enviar las instrucciones que controlen al hardware, envían llamadas al sistema a su hipervisor.

En la paravirtualización, no es necesario simular el hardware para la máquina virtual.

La principal desventaja de la paravirtualización es el hecho de que el S.O invitado debe ser modificado para saber que está siendo virtualizado, y para realizar llamadas al hipervisor, y no directamente al hardware.

**Contenedores**

Es una estrategia completamente distinta a las anteriores. A diferencia de ellas, al emplear contenedores sólo se ejecuta un sistema operativo, que es el mismo para los sistemas anfitrión y huésped.

El sistema operativo anfitrión implementa una serie de medidas para aumentar el grado de separación entre los procesos. Es decir, los procesos que se ejecuten dentro de un contenedor verán un entorno diferente que los demás procesos del S.O. Se busca aislar los procesos de los contenedores.

Dado que el sistema operativo es el único autorizado para tener acceso directo al hardware, no hace falta ejecutar un hipervisor.

Al operar a un nivel más alto, un contenedor presenta algunas limitantes adicionales (principalmente, se pierde la flexibilidad de ejecutar sistemas operativos distintos), pero obtiene también importantes ventajas.

El desarrollo histórico de los contenedores puede rastrearse a la llamada al sistema chroot(), que restringe la visión del sistema de archivos de un proceso a sólo el directorio hacia el cual ésta fue invocada.

Ahora bien, chroot() no es (ni busca ser) un verdadero aislamiento, sólo proporciona un inicio, pero conforme más usuarios comenzaban a utilizarlo para servicios en producción, se hizo claro que resultaría útil ampliar la conveniencia de chroot() a un verdadero aislamiento.

Un punto importante a mencionar cuando se habla de contenedores es que se pierde buena parte de la universalidad mencionada en las secciones anteriores. Si bien las diferentes implementaciones comparten principios básicos de operación, la manera en que logran la separación e incluso la nomenclatura que emplean difieren fuertemente.

El núcleo del sistema crea un grupo para cada contenedor (también conocido como contexto de seguridad), aislándolos entre sí por lo menos en las siguientes áreas:

* **Tablas de procesos:** Los procesos en un sistema Unix se presentan como un árbol, en cuya raíz está siempre el proceso 1, init. Cada contenedor inicia su existencia ejecutando un init propio y enmascarando su identificador de proceso real por el número 1.
* **Señales, comunicación entre procesos:** Ningún proceso de un contenedor debe poder interferir con la ejecución de uno en otro contenedor. El núcleo restringe toda comunicación entre procesos, regiones de memoria compartida y envío de señales entre procesos de distintos grupos.
* **Interfaces de red:** Varía según cada sistema operativo e implementación, pero en líneas generales, cada contenedor tendrá una interfaz de red con una dirección de acceso a medio (\*mac\*) distinta. Claro está, cada una de ellas recibirá una diferente dirección IP, y el núcleo ruteará e incluso aplicará reglas de firewall entre ellas.
* **Dispositivos de hardware:** Normalmente los sistemas huésped no tienen acceso directo a ningún dispositivo en hardware. En algunos casos, el acceso a dispositivos será multiplexado y, en otros, un dispositivo puede especificarse por medio de su configuración. Cabe mencionar que, dado que esta multiplexión no requiere emulación, sino únicamente una cuidadosa planificación, no resulta tan costoso como la emulación.
* **Límites en consumo de recursos:** Casi todas las implementaciones permiten asignar cotas máximas para el consumo de recursos compartidos, como espacio de memoria o disco o tiempo de CPU empleados por cada uno de los contenedores.
* **Nombre del equipo:** Aunque parezca trivial, el nombre con el que una computadora se designa a sí misma debe también ser aislado. Cada contenedor debe poder tener un nombre único e independiente.

Una de las principales características que atrae a muchos administradores a elegir la virtualización por medio de contenedores es un consumo de recursos óptimo: bajo los demás métodos de virtualización (y, particularmente, al hablar de emulación y de virtualización asistida por hardware), una máquina virtual siempre ocupará algunos recursos, así esté inactiva. El hipervisor tendrá que estar notificando a los temporizadores, enviando los paquetes de red recibidos, etc. Bajo un esquema de contenedores, una máquina virtual que no tiene trabajo se convierte sencillamente en un grupo de procesos dormidos, probables candidatos a ser paginados a disco.

**Funciones especiales**

**Suspender e hibernar**

En el mundo real, de las máquinas físicas, cuando suspendemos una computadora, el sistema operativo envía una orden al hardware para que entre en un estado de reposo. El objetivo es reducir el consumo eléctrico lo más que se pueda.

En la suspensión, el contenido de las memorias RAM (y de cualquier memoria principal) **permanece en memoria**. La suspensión reduce el consumo eléctrico, sí, pero la motherboard sigue enviando pulsos eléctricos a las RAM para evitar que los datos se pierdan.

En la hibernación, el objetivo es el mismo (reducir el consumo eléctrico), pero en vez de dejar el contenido de las RAM en… las RAM, el mismo es trasladado a un archivo que se encuentra en disco (hyberfile.sys en Windows, por ejemplo). De esta manera el S.O puede apagar todo el hardware, incluidas las memorias, ya que guardamos todo su contenido en un almacenamiento no volátil.   
  
Cuando suspendemos una VM, no podemos “apagar” el hardware, ya que un hipervisor puede estar manejando a otras varias máquinas virtuales que necesiten de dicho hardware para funcionar. Lo que hace el hipervisor en un S.O virtual al suspenderse una VM es bajar o transferir todo el contenido de la memoria física que está ocupando la VM a un archivo administrado por él con dicho contenido y luego deja a la VM en un estado suspendido, haciendo que ésta ya no demande recursos del hardware subyacente y liberando todos los recursos que ésta utilizaba, los cuales podrán ser utilizados por las demás VMs que quedan en ejecución en la misma máquina física. De esta manera, cuando el hipervisor reanuda la operación (resume) de la VM, realiza la operatoria inversa y transfiere el contenido de dicho archivo a la memoria principal de la máquina física continuando la VM su ejecución desde el punto en que se encontraba al suspenderse.

En la hibernación de una VM sucede algo similar. El sistema operativo guest se encarga de trasladar todo el contenido de su memoria física a un archivo administrado por él y luego envía la señal para apagar su hardware (virtual). Se apaga el hardware virtual de la VM, el hipervisor ve que la VM se apaga, pero este permanece dando servicio de hardware a otras VM.

¿Cuál es la diferencia, si ambas cosas guardan todo en un archivo?

**La diferencia radica en dónde se almacena ese archivo.** En el caso de la suspensión, la operatoria de suspender es llevada a cabo por el hipervisor y el archivo al que se transfiere todo el contenido de la memoria de la VM reside en el hipervisor, es decir, normalmente es un archivo más que se encuentra en el mismo directorio de la VM dentro del File System del hipervisor, y que pasa a ser uno de los tantos archivos que representan a la VM, en este caso una VM suspendida.

En cambio, cuando hibernamos una VM el archivo con el contenido de la memoria de la VM reside o “vive” dentro del File System del sistema operativo guest de la VM. Y yendo un poco más lejos, físicamente, los bloques que ocupa ese contenido pasan a ser bloques que físicamente están dentro del archivo que representa al disco virtual de la VM que se encuentra en el File System del hipervisor (¿por qué? porque el disco de una VM es un disco virtual, que se guarda como un archivo dentro del file system del hipervisor, entonces si escribís algo en el disco de la VM, los cambios se van a ver físicamente en el archivo del hipervisor).

**Clones y templates**

Clonar una VM es tan fácil como entrar al panel de administración del hipervisor y copiar cada uno de los archivos referentes a la máquina virtual. No tenemos que cerciorarnos de tener el mismo hardware en otra PC, ni de todos esos problemas que vienen con la clonación de máquinas físicas. Tan solo copiamos el disco virtual de la VM, los archivos de configuración, y listo, tenemos un clon.

Un template es simplemente un molde de una VM que se utiliza para crear nuevas VMs a partir de él. Es por definición también una VM, nada más que generalmente es una VM que fue instalada una vez y luego no se enciende más, o sólo lo hace para actualizarse y quedar apagada. La idea es que los templates sean lo más genéricos posible: como su nombre lo indica, se trata de que sean plantillas, plantillas que serán usadas para ahorrar tiempo y no tener que configurar una VM desde cero.

En definitiva, clones y templates son casi lo mismo, es crear nuevas VMs a partir de otras. La gran diferencia entre ambos radica en que en los clones, tanto la VM original como su clon continúan su “vida” o ejecución independientemente y cada una se va “ensuciando” con el correr del tiempo típico del uso de cualquier máquina. En cambio con los templates uno siempre tiene el template vacío, “limpio”, recién instalado, ya que el mismo no tiene ejecución, quedó tal cual estaba cuando se lo convirtió en template, y esto permite disponer siempre de una VM base recién instalada para crear nuevas VMs a partir de ella.

Pero si tanto los clones como los templates son una fiel copia de la VM original, ¿esto no acarrea problemas con las nuevas VMs creadas a partir de ellos (pensando en que van a tener los mismos nombres de hosts, las mismas direcciones IP, etc.)? Bueno, los sistemas operativos, como Windows o Linux, que corren dentro de las VMs poseen herramientas para “customizar” los datos que le dan identidad a ellos (como el nombre de host, direcciones IP, dominio al que pertenecen, password de adminstrador, etc.). En el caso de los sistemas operativos Windows esa herramienta es sysprep (de System Preparation). Los hypersores al clonar una VM o al crear una nueva a partir de un template permiten la automatización en la ejecución de dichas herramientas de tal forma que cuando se están creando las nuevas VMs (tanto desde un clon como desde un template) asignarle todos esos nuevos datos a la nueva VM basados en la entrada y opciones seleccionadas por el administrador.

**Snapshot**

Un snapshot es como la palabra lo indica una foto de una VM en un determinado momento. Un snapshot también se lo puede ver como un punto de restauración.  
Los snapshots permiten ser tomados en un determinado momento y luego, de ser necesario, poder volver a ese punto en el tiempo.

Lo que sucede en el hipervisor cuando en una VM se toma un snapshot es que los archivos que representan al disco de la VM en cuestión queda congelado (de hecho, en read-only o sólo lectura). A partir de ese momento, cuando dentro de la VM, en el sistema operativo guest, se realiza una escritura en disco, esa escritura es realizada en el hipervisor en un nuevo archivo que representa ahora al disco de la VM llamado archivo delta o redolog. ¿Por qué ese nombre delta? Porque justamente lo que está guardando a partir de ese momento son todas las modificaciones (el delta, la diferencia) que la VM fue haciendo desde que se tomó el snapshot.

Es importante notar que ahora, el hipervisor tendrá que analizar de donde leer los datos o bloques cada vez que la VM ejecuta un comando de lectura en disco, ya que tiene dos opciones: que el dato leído nunca se modificó desde que se tomó el snapshot, con lo que tendrá que realizar la lectura desde el archivo original que ahora se encuentra en estado de sólo lectura o congelado; o bien si el bloque o dato a leer es un bloque que se modificó luego de que el snapshot de tomó, el bloque deberá ser leído desde el archivo delta.

Hay algunas consideraciones a tomar en cuenta cuenta cuando se utilizan snapshots:

1. Dentro de la VM, el S.O guest nunca se entera de que por “afuera” de él alguien tomó un snapshot.
2. Al sacar snapshots se incurre en una penalización en la performance, ya que ahora por cada comando de lectura que la VM ejecuta el hipervisor debe decernir de qué archivo leer la información.
3. El archivo delta que se genera cuando se toma un snapshot puede crecer hasta un tamaño igual que el archivo base que queda congelado. Por lo tanto, trabajar con snapshots me hace aumentar el espacio en disco utilizado en el hipervisor. Por cada snapshot que se toma en la VM, se está creando en el mismo un archivo delta que potencialmente puede crecer hasta el tamaño del disco congelado.
4. **Los snapshots NO SON UN BACKUP.** Este punto es extremadamente importante ya que hay muchísimos administradores con muchos años de experiencia en la administración de ambientes virtuales que todavía piensan que tomar snapshots en una o varias VMs equivale a tener backups de la misma. Si uno se pone simplemente a analizar el funcionamiento de los snapshots, donde cada vez que se toma un snapshot se congela el archivo base que representa.  
     
   Sin embargo, el uso de snapshots hace posible la toma de backups de las VMs con éstas encendidas. ¿Por qué? Cuando una VM se encuentra encendida el hipervisor mantiene un bloqueo o lock sobre los archivos que representan al disco virtual de la VM. Esto es debido a que el hipervisor está escribiendo en dicho archivo todo lo que esa VM envía a escribir en su disco dentro de la misma. Si uno quisiera tomar un backup de la VM copiando desde el hipervisor los archivos que la representan, no podría porque existe ese bloqueo del hipervisor sobre los mencionados archivos que me imposibilita leerlos desde otra aplicación. En cambio, si antes de tomar el backup uno le saca un snapshot a la misma, los archivos que representan a los discos virtuales quedan liberados del bloqueo ya que quedan en modo de sólo lectura. Es ahí donde entra en juego la aplicación de backup que ahora sí puede leer completamente esos archivos y guardarlos en una ubicación segura. Por lo tanto, los snapshots no son backups de las VMs, pero sí son una herramienta o funcionalidad que me permiten tomar backups.

¿Qué sucede cuando uno quiere hacer uso de la función de volver el tiempo atrás al momento de haber tomado el snapshot? Simplemente lo que hace el hipervisor es descartar todos los cambios que se grabaron en el archivo delta (eliminando el archivo) y volver a trabajar directamente sobre el archivo base, restaurando el bloqueo o lock sobre el mismo y haciendo que nuevamente las escrituras de la VM se realicen sobre el mismo.  
  
¿Y qué sucede por otro lado si lo que se quiere es mantener el estado de la VM después de haber tomado un snapshot pero quiere descartar al snapshot en sí? Es decir, se pretende continuar con la ejecución de la VM en forma normal sin snapshot alguno. Lo que hace el hipervisor es hacer un “commit” o volcado de todos los datos que se encuentran en el archivo delta (que mantiene todos los cambios luego del snapshot y que son cambios que no se quieren perder) al archivo base.   
  
Generalmente, los hipervisores toman el hecho de borrar una snapshot como el hecho de hacer un commit y volcar todos los cambios.

**Migración**

La migración de las máquinas virtuales puede involucrar por un lado la migración de los archivos que la representan desde una ubicación a otra en el hipervisor y por el otro la migración de los procesos que la representan de un hipervisor a otro en diferentes máquinas físicas. Es importante aclarar que uno no implica el otro. Es decir, son dos tipos de migraciones por separado e independientes entre sí y que generalmente se realizan por separado.

La VM puede estar apagada, encendida o suspendida durante una migración. Las migraciones con la VM apagada se conocen como “cold migration” y con la VM prendida como migración en caliente o “live migration”.

**Problemáticas de las migraciones**

**Migraciones en frío**

Este tipo de migraciones no representan mayores problemas. Simplemente se trata de migraciones de los archivos que representan a la VM a migrar de una ubicación del hipervisor a otra. Nótese que dicha ubicación de archivos puede ser local al hypervisor (discos locales residentes en la máquina física donde se ejecuta el hipervisor) o remota (archivos accedidos por el hipervisor a través de una red SAN como ya se vio en el módulo de entrada/salida en el apartado de tecnologías de almacenamiento).   
  
Estas migraciones pueden entonces ser llevadas a cabo entre diferentes ubicaciones locales, remotas o entre ambas. Por ejemplo, se podría querer migrar una VM que reside en un disco local del hipervisor a un volumen de datos que se encuentra en un sistema de almacenamiento externo accedido por Fibre Channel.

**Migraciones en caliente**

Estas son las migraciones más complejas a llevarse a cabo. Las migraciones que se pueden realizar en este estado son de archivos (entre dos ubicaciones accesibles al hipervisor) o de procesos (los procesos que representan a la VM migran de una máquina física a otra). Es importante aclarar aquí para que no haya confusiones que cuando se dice que se migran los procesos que representan a una VM de una máquina física a otra se está migrando la VM completa, es decir, se migra la ejecución de una VM que se está ejecutando en el servidor A al servidor B sin que haya tiempo sin servicio (downtime). No debe bajo ningún punto de vista entenderse esta migración como que se migran los procesos que se ejecutan dentro del sistema operativo guest a otro sistema operativo guest en otra VM en otra máquina física. Lo que se migra es la VM completa.   
  
**Migración de archivos:** En este caso teniendo la VM encendida el hipervisor inicia una migración de los archivos que representan a la VM de una ubicación en su File System a otra (recordemos que dicho File System puede hacer referencia a archivos locales o remotos). El problema es que mientras se están copiando los archivos, la VM sigue realizando operaciones de E/S, pudiendo modificar el disco virtual de la VM. Esto presenta un problema, ya que los archivos de la máquina virtual sufrirán cambios y esos cambios tienen que verse reflejados.  
  
La primera solución o acercamiento a solucionar este problema fue tomar snapshots del archivo, y tener un archivo de diferencias, con el objetivo de ir pasando gradualmente dichas diferencias… El problema con este acercamiento es que, si el archivo que estás copiando es uno que sufre constantemente modificaciones, el archivo de diferencias va a ser gigante, y nunca vas a terminar de copiarlo. Estarías perdiendo tiempo y no garantizarías tener 100% uptime.  
  
En la actualidad, el hipervisor al iniciar una migración de los archivos de una VM comienza directamente a copiar todos los bloques del archivo que representa al disco rígido virtual. Mientras tanto la VM continúa su ejecución, ejecutando lecturas y escrituras sobre su disco. Si la VM ejecuta una escritura en un bloque que aún no se copió por debajo en el hipervisor a la nueva ubicación, dicha escritura se escribe directamente en el archivo original, ya que en algún momento el hipervisor llegará hasta dicho bloque y lo copiará al destino. En cambio si la VM escribe un bloque que ya fue escrito por el hipervisor a la nueva ubicación, entonces el sistema “divide” la escritura en dos (se dice que se realiza un “Split” del IO) y se generan dos comandos de escritura con igual contenido, pero uno se escribe en el archivo original y el otro en el archivo de destino.  
  
**Migración del proceso de la VM*:***El proceso generalmente se basa en comenzar a migrar todo el estado de ejecución de la VM a migrarse a través de la red dedicada para tal fin y, como la VM sigue en producción y funcionando, un proceso en el hipervisor va tomando cuenta de todas las páginas de memoria que se van modificando (formando una especie de archivo de bitmap) mientras la migración se lleva a cabo. Cuando la primera fase de la migración de datos se completa se freeza la VM temporalmente para copiar las páginas modificadas y se reanuda por último la ejecución desde la siguiente instrucción en el servidor de destino.

* Primer punto a observar: Para que una nueva máquina física pueda continuar con la ejecución de una VM que venía ejecutándose en otro lado, la nueva máquina física tiene que tener acceso sí o sí a los archivos de la VM. Si no, ¿desde dónde leerá o escribirá cuando el SO guest realice operaciones de E/S? Por lo tanto una de las primeras condiciones que se tienen que cumplir para que se pueda llevar a cabo la migración de una VM de un servidor físico a otro es que los archivos de la VM en cuestión se encuentren en un espacio de almacenamiento compartido, es decir, una SAN. Si bien ahora algunos hipervisores permiten la migración de la ejecución de una VM que se encuentra en discos locales a otro servidor, lo que hacen es primero una migración de los archivos de dicha VM al disco local del servidor de destino y luego realizan la migración de la ejecución.
* Segundo punto. Cuando una VM migra a un nuevo servidor físico, la misma pretende seguir ejecutando de la misma manera que lo venía haciendo. En realidad, la VM nunca se entera ni que es una VM ni que se migró su ejecución. Por eso es que ella no hará ningún cambio en su ejecución y deberán estar dadas las condiciones para que ello se pueda cumplir.

Con seguir ejecutando de la misma manera nos referimos a que la VM seguirá ejecutando las mismas instrucciones que venía ejecutando antes de ser migrada. Por lo tanto, deberá haber una compatibilidad entre las CPU del servidor físico de origen y el de destino.   
  
Ahora bien, ¿qué significa que dos CPUs sean compatibles o no? Dependiendo del hipervisor estas restricciones de compatibilidad serán más laxas en algunos y más estrictos en otro. Generalmente las características que no implican incompatibilidad son la velocidad de las CPUs y los tamaños de las cache internas. En cambio dentro de las características que definen la compatibilidad o no de dos CPUs se encuentran la marca del procesador (generalmente AMD e Intel no son compatibles para migraciones en caliente), set de instrucciones y soporte o no de virtualización asistida por hardware.

* Tercer punto. Si una VM está utilizando algún dispositivo de hardware físico del servidor donde se encuentra ejecutando, dicha VM generalmente tendrá imposibilitada la opción de ser migrada a otro servidor físico, ya que en el servidor de destino no se tendrá acceso directo al hardware físico de otro servidor. Cuando se utilizan este tipo de configuraciones, esos usos del hardware directo actúan como si fuera un ancla que le imposibilita la migración fuera de ese servidor específico. El uso más común de dispositivos físicos para una VM viene dado cuando se le configura a la misma que utilice el CDROM del servidor físico. Cuando esto ocurre, generalmente el administrador de la infraestructura virtual elimina el uso de dichos dispositivos (si es que puede hacerlo con la VM encendida) para posibilitar así la migración.

**Migraciones en estado suspendido**

Si bien cuando una VM suspendida está apagada, el estado real es “suspendido”.

¿Por qué merece un análisis la migración de las VMs en este estado si la VM está apagada? ¿No se podría tratar a la misma como una migración en frío? En realidad la respuesta es no.

Pensemos que la VM se suspendió y cuando se encienda nuevamente (operación de “resume”) la misma va a pretender continuar con la tarea que estaba realizando en ese momento. La VM no vuelve a bootear y “redescubrir” en qué procesador está ejecutando para saber qué set de instrucciones está disponible, sino que va a continuar trabajando con el mismo que estaba utilizando al momento de suspenderse. Por tanto, para la migración de la VM de un servidor físico a otro se tendrán que cumplir las mismas condiciones que para la migración en caliente.

**Aprovisionamiento delgado o Thin Provisioning**

Existen básicamente dos modos o tipos de aprovisionamiento: el *aprovisionamiento gordo, grueso, o thick provisioning*; y el *aprovisionamiento delgado, o thin provisioning.*

En el *thick provisioning* lo que ocurre es que un administrador de una infraestructura virtual decide crear una VM y le asigna un disco de determinado tamaño e inmediatamente en el hipervisor se crear un archivo de ese mismo tamaño. No importa si el sistema operativo guest dentro de la VM escriba o no dato alguno. Siquiera importa si en la VM incluso se ha instalado o no un sistema operativo guest, ya que el hipervisor automáticamente crea un archivo del tamaño del disco virtual inmediatamente cuando la VM es creada o el nuevo disco es agregado a una VM ya existente.

En el *aprovisionamiento delgado o thin provisioning* sin embargo ocurre otra cosa. Cuando la VM es creada y asignada a la misma un disco, el hipervisor inmediatamente crea el archivo que representa a ese disco. Sin embargo, en un principio dicho archivo tiene tamaño cero (ya que dentro de la VM todavía no se ha escrito ningún dato en dicho disco). A medida que la VM vaya escribiendo datos dentro de ese disco, entonces esos datos se irán escribiendo en el archivo subyacente del hipervisor y por lo tanto el mismo irá creciendo bajo demanda. Esto quiere decir que cuando la VM escriba su primer Giga Byte, el archivo tendrá 1 Gb, y a medida que se vayan escribiendo más y más datos el tamaño del archivo en el hipervisor irá creciendo. Por eso se dice que el tamaño de dicho archivo crece bajo demanda, porque cuando se demanda más espacio el hipervisor le asigna los bloques de disco necesarios a dicho archivo.

Lo que tiene que quedar en claro de ambos tipos de aprovisionamiento es que el sistema operativo guest SIEMPRE “ve” un disco rígido del tamaño aprovisionado, en nuestro ejemplo un disco de 50 GB. Es decir, si se utiliza thin provisioning para asignarle a una VM un disco de 50 GB, el sistema operativo guest de la VM ve que tiene un disco de 50 GB, no importa si la ocupación de ese disco es de 20 GB y por ende, en el hipervisor el archivo que representa a dicho disco también ocupa 20 GB. El guest siempre ve que tiene un disco de 50 GB.

La funcionalidad de thin provisioning es una herramienta extremadamente útil en los sistemas virtuales, porque nos permiten ir consumiendo el espacio físico a medida que realmente lo necesitamos, pero mal administrado o mal utilizado se puede convertir en un verdadero problema. Por eso en cuanto comenzamos a utilizar esta funcionalidad debemos prestar constante atención al espacio físico utilizado y disponible. Es por esto que las consolas de administración de este tipo de sistemas poseen la función de crear alarmas que permiten monitorear tanto el espacio libre disponible como el espacio aprovisionado. El espacio aprovisionado es el espacio que se “prometió”. Por lo tanto, cuando el espacio aprovisionado pasa del 100% del espacio físico, comienza el riesgo de quedarnos sin espacio. Cuanto más allá del 100% se encuentre este espacio, más riesgo habrá de quedarnos sin espacio.

**Storage**

El storage hace referencia a todos aquellos dispositivos y equipos de almacenamiento externo que sirven para guardar una cantidad masiva de datos.   
  
El storage es la forma en que los servidores y los centros de datos pueden guardar y almacenar toda su información. Debido al gran volumen de datos que uno o más servidores pueden manejar, es necesario tener dispositivos y tecnologías que nos faciliten el manejo de los mismos.

Los storage nos permiten alojar en los dispositivos de almacenamiento los discos, en uno o varios chasis y los diferentes servidores acceden a la información como si estuvieran accediendo directamente a discos conectados internamente a ellos. Una de las ventajas de esto, es que si un servidor queda fuera de servicio, la información no se ve afectada porque es totalmente independiente de los servidores. Como desventaja, ahora el único punto de falla serían los dispositivos de almacenamiento… pero para eso existen los RAIDS y otras técnicas de redundancia y protección de datos.

**Conceptos asociados a storage**

* **SAN (Storage Area Network):** La SAN es una red de almacenamiento que conecta a los servidores con los storage. Según el tipo de tecnología utilizada (tipo de protocolo) la SAN puede ser una red totalmente separada físicamente de la red LAN que todos conocemos o bien, compartir la red LAN y quedar separada “lógicamente”.
* **HBA (Host Bus Adapter)**: Genéricamente hablando, la HBA es la interfaz con la que se conectan los servidores a la red SAN para llegar a los storage. Otra vez, según el tipo de protocolo y tecnología esta HBA podrá ser una placa física que se conecta en el servidor (por ejemplo las HBA para Fibre Channel que veremos a continuación), una placa de red común y corriente (en el caso de los protocolos de almacenamiento que comparten la red LAN) o por último, una CNA.
* **CNA (Converged Network Adapter):** En las redes convergentes de datos se envían al mismo tiempo tráfico de red común y corriente y tráfico de almacenamiento. Estas placas generalmente se presentan al sistema operativo como placas de red y al mismo tiempo como placas HBA para almacenamiento. Es decir, una misma placa física, dentro del sistema operativo se ve como si fueran dos. Estas placas manejan el protocolo de almacenamiento FCoE.
* **Controladoras (o Storage Processors o simplemente SPs):** Son las encargadas de llevar a cabo toda la operatoria dentro de los storage. Son las que tienen la “inteligencia” de estos sistemas. La mayoría de éstos vienen en una configuración dual-controller, lo que significa que tienen dos controladoras para brindar redundancia. Estas controladoras, al igual que lo hacen las HBA, también se conectan a la SAN para quedar de esta manera conectados con los servidores. Según el storage, las controladoras pueden trabajar en modo activo-activo o activo-pasivo. En este último caso, la controladora pasiva o standby sólo entra en servicio cuando la primera falla. En cambio en los storage activo-activo ambas se encuentran dando servicio al mismo tiempo, repartiendo la carga entre las mismas, además de ofrecer, claro está, capacidad de redundancia o failover.
* **Latencia:** En redes la latencia es sinónimo de demora. En el ambiente de almacenamiento hay muchísimas latencias a medir (latencia interna del kernel del sistema operativo, que mide cuánto tiempo el comando SCSI generado por el SO tarda en entregarse a la HBA, latencia interna de la HBA, que mide cuánto tiempo tarda el comando SCSI en salir al medio físico, etc.). La latencia que más nos interesa generalmente es la latencia física total, es decir, el tiempo que tarda desde que se genera un comando SCSI dentro del sistema operativo hasta que llega la respuesta por parte del storage (es decir, sumando la latencia interna del SO, la de la HBA, el tiempo que tarda en transmitirse el comando por la red física y todo su camino de vuelta).

**Protocolos de comunicación**

**Orientados a bloques**

**def. SCSI:** Small Computer System Interface,pequeña interfaz del sistema de cómputo. SCSI es un conjunto de estándares para conectar físicamente y transferir datos entre computadoras y equipos periféricos. Los estándares SCSI definen comandos, protocolos, e interfaces eléctricas, ópticas, y lógicas. Las computadoras y los equipos de storage utilizan un modelo de cliente-servidor para comunicarse. La computadora es un cliente que realiza peticiones al storage, por ejemplo, para escribir o leer datos.

En los protocolos orientados a bloques, cuando los servidores quieren acceder a algún dato simplemente envían al storage un comando SCSI indicando el número de bloque a leer/escribir. Este comando viaja hacia el storage, alguna de las controladoras del storage lo agarra, va al o a los discos donde se encuentra dicho bloque y realiza la operación. A la vuelta, el storage le contesta al servidor con otro comando SCSI indicándole la respuesta al comando original.

El tema es cómo viaja ese comando. Para eso se desarrollaron los diferentes protocolos que se detallan a continuación (los más utilizados):

* **FC o Fibre Channel:** Básicamente el comando SCSI se encapsula en un frame Fibre Channel y dicho frame se entrega a la HBA del servidor y la HBA físicamente (generalmente con haces de luz porque se utiliza fibra) hace llegar el comando al storage. La vuelta es similar. Lo importante acá es que cuando tenemos dos servidores no hay problema porque conectamos las HBAs directamente a las controladoras del storage (generalmente cada controladora tiene dos puertos por lo que permite que dos servidores se conecten en forma redundante). El problema aparece cuando tenemos un tercero, cuarto, quinto, etc. ¿Qué hacemos? Tenemos que poner algo en el medio, y ese algo es un switch FC, que al igual que los switches ethernet, se utilizan para concentrar la red en algún punto y hacer que varios servidores se puedan conectar con el mismo storage. A esos switches en la jerga de FC se los conoce como “Fabric”.

Al igual que en las redes LAN había una manera de segmentar dicha red en forma lógica, utilizando las VLANs (redes virtuales) en las redes Fibre Channel existen el concepto de “zonning” o zonificación. El zonning permite que se definan zonas de trabajo y hace que solamente los equipos que pertenecen a la misma zona se puedan ver entre ellos. De esta manera, aunque los switches sean los mismos, mediante el uso de las zonas puedo configurar que ciertos servidores vean determinados storage y otros servidores vean a otros storage, dando seguridad al sistema y evitando que un equipo vea a otro que no tiene por qué ver.

* **iSCSI o IP SCSI**. Los mismos comandos SCSI del punto anterior en lugar de encapsularse en un frame Fibre Channel se pueden encapsular en un segmento TCP, después en un paquete IP y después en un frame Ethernet. ¿Con qué sentido? Si tenemos un frame ethernet podemos utilizar los mismos switches ethernet que todos ya tenemos. Eso es iSCSI.   
    
  Ahora la SAN (Storage Are Network) la armo utilizando los mismos equipos que ya vengo utilizando para interconectar a las computadoras y servidores de mi red. ¿Tengo que tener algo en cuenta? Debería. Por ejemplo debería considerar seriamente dedicar una VLAN (Virtual LAN) sólo para el tráfico iSCSI, de tal manera que los broadcast que se generan en mi red no me perjudiquen a los frames iSCSI (no se olviden que esto está simulando una SAN y la latencia la quiero mantener lo más baja posible). Por lo tanto, si utilizo la misma red física para transportar el tráfico de red común y corriente y el tráfico de almacenamiento, al menos tengo que separar lógicamente una red de otra (por eso la recomendación del uso de VLANs). Por otro lado debería considerar habilitar los Jumbo Frames, frames que exceden el tamaño estándar de una red LAN (1500 Bytes) de manera tal de bajar lo más posible el overhead creado por el múltiple encapsulamiento en las diferentes capas.
* **FCoE o Fibre Channel Over Ethernet**. A medida que avanzaba el tiempo, otro equipo de trabajo pensó que en lugar de pasar por la capa de transporte y la capa de red, podíamos agarrar el mismo comando SCSI y encapsularlo en un frame Fibre Channel, luego en un Frame Ethernet y seguir utilizando switches ethernet de red. Sin embargo ahora no me es posible seguir utilizando los mismos switches de red que venía utilizando anteriormente. Esto es debido a que Fibre Channel no es un protocolo que esté preparado para perder tramas (frames). En cambio Ethernet es lo que se denomina un protocolo de mejor esfuerzo, es decir, hago lo mejor posible para que las tramas lleguen a su destino, pero si alguna se pierde, alguien tendrá que retransmitir la información. ¿Por qué entonces puedo utilizar iSCSI que a su vez utiliza los switches ethernet comúnes y corriente y no lo puedo hacer con FCoE? Sucede que iSCSI en la primera capa en que se encapsula es la de transporte, utilizando el protocolo TCP. En el mundo del TCP/IP, este trabajo de control sobre los segmentos lo realiza justamente el TCP. Sin embargo, FCoE no utiliza TCP (como sí lo hacía iSCSI), por lo tanto, ¿quién es el que me va a brindar seguridad? Si no lo hacen los protocolos, entonces lo debe hacer el hardware. Ergo, para poder utilizar FCoE se debe contar con switches Ethernet especiales que utilizan la tecnología de Data Center Bridging, la cual permite asegurar que el tráfico Fibre Channel que viene encapsulado en Ethernet llegue sin problemas a destino, sin pérdida de frames. A estos switches de los conoce con el nombre de Lostless (sin pérdida).
* **SAS**. Éste es un híbrido entre la tecnología SAN y Direct-Attach (es decir, discos conectados directamente en el servidor). No es ni una SAN pura ni tampoco son discos directamente conectados (se comporta como una SAN, pero la tecnología que se utiliza es la de los discos conectados directamente). Sólo van a ver sistemas de storage con conectividad SAS en IBM, HP y LSI. ¿Qué es lo que hicieron estas marcas? Agarraron el bus interno SAS que se utiliza dentro del servidor y lo extendieron fuera del mismo. Pusieron mecanismos de arbitraje dentro de las controladoras para que varios servidores pudieran acceder a los datos al mismo tiempo (sin “chocar” entre ellos) y lograron utilizar la velocidad SAS de 3Gbps a un costo muy accesible. Lo bueno es que se obtiene muy buena performance a un costo accesible. Lo malo es que las controladoras tienen cuatro puertos cada uno. Pasa lo mismo que con Fibre Channel pero con 4 servidores. Cuando queremos poner el quinto servidor hay que poner un switch. El problema es que no son muy comunes estos swithes. Donde se los ve más a menudo es en los servidores de tipo blades. Como decíamos anteriormente, a estas redes con estos storage no se las considera una SAN en el sentido más puro de la definición (ya que no utilizan ninguno de los protocolos mencionados anteriormente) pero brindan el mismo servicio como si lo fueran (es decir, permiten que varios servidores accedan a un mismo storage al mismo tiempo). Por eso decimos que son un híbrido. Se las considera direct attach.

**Protocolos orientados a archivos**

En lugar de que el servidor ejecute un comando SCSI sobre el storage lo que hace es hacer una consulta de un determinado archivo en una determinada ruta. A los equipos que trabajan en esta modalidad se los conoce como NAS (Network Attach Storage). Los dos protocolos más conocidos son:

* **NFS:** Network File System. Desarrollado por Sun Microsystems para su sistema SUN Solaris. Se convirtió en un estándar para la compartición de archivos en sistemas UNIX. En la jerga de NFS el directorio a compartir se dice que se “exporta”.
* **CIFS:** Common Internet File System. Cada vez que un usuario en Windows ingresa en el cuadro de texto para ejecutar \\dirección\_IP\_Servidor en Windows para acceder a una carpeta compartida que se encuentra en otra máquina, están haciendo uso de este protocolo. Es también conocido como Samba o SMB. Es el protocolo desarrollado por Microsoft para la compartición de archivos e impresoras en sistemas Windows y también se lo conoce como Samba porque es el *daemon* que se levanta en sistemas Linux para acceder a carpetas compartidas en Windows.

**Orientado a bloques vs orientado a archivos**

La diferencia radica en quién es el que ejecuta el comando SCSI y qué es lo que viaja por la red. En los protocolos orientados a bloques, el sistema operativo que se ejecuta en el servidor “ve” a los discos que están en el storage como si tuvieran acceso directo a ellos y ejecutan directamente los comandos SCSI. Es decir, en los protocolos orientados a bloques los storage le presentan a los servidores unidades lógicas (llamadas LUN – Logical Unit Number) que los servidores “ven” como discos propios. Y por lo tanto, los servidores ejecutan los comandos SCSI sobre estas LUNs directamente, y estos comandos son los que viajan por la red.

En los protocolos orientados a archivos la información que viaja por la red como peticiones son comandos del propio protocolo (comandos definidos en NFS o en CIFS) con sus diferentes parámetros, entre ellos la ruta y el archivo a acceder. Es decir, el sistema operativo que se ejecuta sobre el servidor sabe que la información a la que está accediendo es un directorio exportado o una carpeta compartida, según el protocolo, y envía las órdenes para acceder a ellos a la controladora del storage, y es esta última la que ejecuta el comando SCSI para acceder a la información.

**Conceptos importantes sobre RAID**

Antes de hablar de los diferentes niveles de RAID, definamos algunos conceptos:

* **Discos físicos:** Son los discos físicos que todos conocemos.
* **Arreglo o Array físico:** Es el conjunto de discos físicos. Uno o más discos físicos unidos conforman un arreglo o array físico de discos.
* **Arreglo o Array lógico:** Los arreglos o array lógicos se conforman particionando un array físico o bien combinando más de un array físico en un array lógico. Por ejemplo, se podría tener un arreglo físico de 10 discos físicos, y a partir de él crear dos arreglos lógicos conformados cada uno por ejemplo, de 5 discos físicos cada uno. Así también se podría tener dos array físicos de 3 discos físicos cada uno, y formar a partir de ellos un solo array lógico conformado por la totalidad de los 6 discos. En la gran mayoría de los casos prácticos, esta relación es de 1 a 1, es decir, se crea un arreglo lógico por cada arreglo físico.
* **Discos lógicos:** Los discos lógicos se crean dentro de los array lógicos y se pueden crear más de un disco lógico por cada array lógico. En definitiva, estos discos son los que “ve” el sistema operativo como uno solo. Estos discos lógicos en la mayoría de los sistemas de almacenamiento se los conoce como LUN (Logical Unit Number).

**Mirroring**

El sistema escribe en dos discos la misma información, se espeja. El mirroring provee excelente redundancia ante fallos, y además en caso de que falle un disco, la recuperación es instantánea (se accede al disco que no falló inmediatamente).

La desventaja es que al espejar, estamos perdiendo un 50% de capacidad, ya que el otro disco solo va a guardar la misma información que tenemos en el primero.

**Duplexing**

Similar al concepto de mirroring, solo que además también se duplica la controladora de los discos. Esto provee redundancia en caso de que la controladora falle. Las controladoras pueden trabajar en modo activo/pasivo (solo una trabaja, la otra cambia de estado en caso de fallo) o activa/activa (ambas trabajan, y esto permite a los sistemas enviarle

las peticiones de entrada/salida (IO) a ambas controladoras a la vez, permitiendo así un reparto de carga entre ambas controladoras).

**Paridad**

La paridad es una técnica que nos asegura una redundancia en caso de errores, sin las desventajas en las que ocurre el mirrroring (principalmente la pérdida de espacio). La paridad se basa en técnicas de control de errores, como los códigos de Hamming, o en el uso de compuertas XOR.

Lo mejor de usar compuertas XOR es que si uno calcula por ejemplo A XOR B, nos arroja un resultado que, si a su vez lo volvemos a operar con la misma operación con cualquiera de los dos operandos, nos da como resultado el otro operando. Por ejemplo:

A XOR B = C

Si ahora hacemos:

C XOR B = A

O bien:

C XOR A = B

De esta forma podemos reconstruir los datos faltantes en caso de que falle un disco.

Esta técnica de utilizar la paridad en conjunto con el striping me permite ahora disponer de redundancia en los datos frente a una eventual falla en alguno de los discos y además mejorar mi performance porque ahora escribo en más de un disco al mismo tiempo.

Pero por otro lado ahora tengo los siguientes problemas. Ahora cuando escribo un dato en el array, tengo que hacer un cálculo de paridad que antes con el mirroring no era necesario. Por eso, es muy importante disponer de controladoras de alta capacidad de cómputo ya que el retardo introducido por dicho cálculo impactará directamente en la performance.

Además, como contra tiene que la recuperación ante fallos ya no es tan transparente como en la técnica de mirroring, ya que si el disco que falla NO es el de paridad, ahora cada vez que tenga que recuperar un valor del disco que falló tengo que también hacer el cálculo de paridad.

**Performance en ambientes RAID**

Los RAID utilizan paralelismo (acceso a muchos discos en paralelo) para poder brindar mejor rendimiento que un solo disco individual.

**Mirroring:** leer es más rápido que escribir. Al escribir tenemos que escribir dos veces la misma información en dos discos diferentes. Al leer, un disco puede responder la petición de lectura, mientras que el otro responde a otra diferente.

**Striping sin redundancia:** tanto la lectura como la escritura tienen la misma tasa de transferencia, aproximadamente.

**Striping con paridad:** la escritura es peor que la lectura, pero la penalidad en escritura es más sustancial.

* **Para las lecturas:** la lectura es más rápida ya que no hay necesidad de leer la información de paridad cuando llega una petición.
* **Para escrituras secuenciales:** hay un doble overhead, ya que se tiene que calcular la paridad, y escribir dicha paridad en un disco.
* **Para lecturas aleatorias:** el paralelismo nos garantiza lecturas veloces. Unos discos pueden estar leyendo partes de un stripe de una petición, mientras que otros discos pueden estar leyendo partes del stripe de otra petición. No hace falta leer la paridad.
* **Para escrituras aleatorias:** imagínate que tenés este documento abierto y decidís modificarlo. Esa modificación va a cambiar los bits del archivo en el disco. Si esos bits cambian, la paridad también debe cambiar para reflejar el nuevo estado. Esto incurre en una gran penalidad para las escrituras aleatorias: no solo tenemos que escribir un nuevo valor en disco, si no que tenemos que calcular nuevamente la paridad.   
    
  Algunos niveles de RAID leen nuevamente todos los bloques pertenecientes al archivo… y calculan la paridad en base a esto. Es altamente ineficiente esta forma.  
    
  Otros, como RAID 5, utilizan un poco la magia de las XOR y solamente leen los datos que se van a cambiar y la paridad vieja, y a partir de ahí escriben los nuevos datos y escriben la nueva paridad.   
    
  Sin embargo, por estos motivos, no es recomendable utilizar algo como RAID 5 cuando tenemos escrituras aleatorias frecuentes (como en el caso de una base de datos).  
    
  Ni hablar de los RAIDs 3 o 4, en donde tenemos un disco donde se guardan los datos de paridad. Ese disco genera un cuello de botella.  
    
  Además, la escritura debe realizarse sin interrupciones. La controladora debe asegurarse de escribir todos los cambios y calcular la paridad sin interrumpir el proceso, de otra forma, la paridad no podría reflejar el verdadero contenido del archivo.  
    
  De todo este desarrollo, podemos sacar como conclusión que la técnica de paridad junto con striping es la más performante de todas siempre y cuando se realice una escritura completa del stripe. Sin embargo, es un hecho que en la mayoría de los ambientes modernos se utiliza la virtualización, y en éstos la mayoría de las escrituras son pequeñas y aleatorias, con lo que es muy difícil que se escriban stripes enteros.  
    
  Entonces, ¿para qué sirve esta técnica de escribir todo el stripe al mismo tiempo si en la mayoría de los casos no la voy a poder utilizar? Acá es donde viene la ayuda de las memorias cache. Si estas memorias se utilizan para la escritura, el storage puede demorar la escritura de los diferentes segmentos hasta que los mismos conformen un stripe entero y recién ahí volcar físicamente los datos a los discos.

**Performance asociadas al posicionamiento y a la transferencia**

* **Tiempo de posicionamiento:** Aquel tiempo que se emplea para que la cabeza del disco se sitúe en la posición adecuada para leer o escribir un dato.
* **Tiempo de transferencia:** Aquel tiempo que se emplea para transferir un dato desde o hacia el disco una vez que la cabeza está posicionada.

**Ancho y tamaño del stripe**

Hay dos parámetros con que los sistemas de RAID permiten trabajar. Uno es el ancho del stripe (stripe width) y el otro es el tamaño (stripe size).

El ancho del stripe se define como la cantidad de segmentos que se pueden escribir o leer simultáneamente de los discos, y como es lógico, concuerda con la cantidad de discos. Es decir, si tenemos 4 discos, el ancho de nuestro stripe será de 4. A mayor ancho de stripe mayor performance, ya que se aumenta el paralelismo.

El segundo parámetro, stripe size, también es conocido como block size, chunk size, stripe length o granularity. Se refiere al tamaño de cada uno de los segmentos que se escriben o leen en un disco. Este tamaño sí es configurable por el usuario y típicamente se configura entre los 2 Kb y los 512 Kb en potencias de 2.

El impacto en la performance según los diferentes tamaños del stripe es más difícil de cuantificar que con el ancho:

* **Disminuir el tamaño del stripe:** Al disminuir el tamaño del stripe se hace que los archivos se dividan en partes más pequeñas y utilicen más discos, disminuyendo así el tiempo de transferencia, pero aumentando el de posicionamiento.
* **Aumentar el tamaño del stripe:** Permite que los archivos entren en menos bloques, disminuyendo el tiempo de posicionamiento y aumentando el de transferencia. El tiempo de posicionamiento se ve mejorado en caso que la controladora tenga la inteligencia suficiente para atender otro requerimiento en discos que no se utilicen dentro del array.

No hay una receta que indique cuál es el mejor tamaño. Todo depende. Si realmente uno quiere asegurarse de cuál es el mejor tamaño, lo recomendable es hacer pruebas y estudios, cambiando los tamaños y viendo cuáles tienen mejor rendimiento para el uso que se les está dando.

**Operación degradada y reconstrucción**

Cuando se rompe un disco, el sistema entra en Operación Degradada. Se dice que la operación está degradada porque ahora el sistema tiene un disco menos (o dos según el nivel de RAID) y el hardware que queda funcionando tiene que compensar esta pérdida.

Por otro lado, un gran detractor de la perfomance en estos casos es la reconstrucción de datos, que toma lugar cuando se reemplaza el disco fallado con uno nuevo. Esta reconstrucción, que puede llegar a tomar horas, es el proceso por el cual el sistema tiene que volver a “llenar” el disco nuevo de reemplazo con los datos que se encontraban en el que falló. Esta operatoria puede ser sencillamente copiar los datos del disco que quedó sano en el nuevo (en los sistemas que utilizar mirroring) como tan compleja como tener que calcular todas las paridades en el disco nuevo (en los sistemas que utilizan striping con paridad).

Los sistemas actuales de almacenamiento permiten que el administrador elija si esa reconstrucción se hará automáticamente, en caso que el sistema disponga de lo que se llama disco de *spare (repuesto)*, donde el administrador deja uno de todos los discos fuera de línea (off-line) y el mismo entra en juego cuando uno perteneciente a algún array en el sistema se rompe; o bien se hará manualmente, donde el administrador elige el momento en que se hará la reconstrucción. Debido a que la pérdida de performance es generalmente considerable, muchas veces no es mala idea esperar a horarios de baja carga transaccional para realizar el trabajo, no obstante teniendo en cuenta que en la mayoría de las configuraciones cuando un disco se rompe ya no queda lugar a la rotura de un segundo disco sin que esto lleve a la pérdida de información.

**Niveles de RAID**

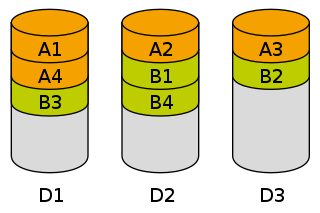
Los discos suelen ser la parte que más se rompe dentro de las computadoras. También suelen ser las partes más lentas de las mismas. Para solucionar estos problemas, en vez de utilizar un solo disco de gran tamaño, se empezaron a utilizar varios discos pequeños en paralelo. De esta forma se mejora el ancho de banda, y se pueden implementar técnicas de redundancia.

RAID (Redundant Array of Inexpensive/Independent Disks) es una configuración hecha para la redundancia, que se puede implementar en varios niveles.

**RAID nivel 0**

Para entender el RAID 0, primero hay que entender en qué consiste el **Data Striping**.

El Data Striping es una técnica donde datos secuenciales (por ejemplo, un archivo) son segmentados lógicamente, pudiendo lograr que los segmentos consecutivos sean almacenados en diferentes medios físicos.



Ejemplo de data striping. Los archivos A y B, de cuatros bloques cada uno,

están repartidos entre los discos D1 a D3.

Teniendo en cuenta esto, ahora podemos definir el RAID 0.

Un RAID 0 distribuye los datos equitativamente entre dos o más discos utilizando data striping. Es importante recalcar que **este nivel de** **RAID no usa redundancia**, es decir, los datos no son repetidos, ni tampoco se implementan controles para asegurarlos ante una falla.

Este nivel de RAID se usa para proporcionar un alto rendimiento de escritura ya que los datos se escriben en uno o más discos de forma paralela, aunque el archivo solo esté presente una vez en todo el conjunto de discos. También puede ser usado para dar la ilusión de un disco virtual grande, creado a partir de discos pequeños.

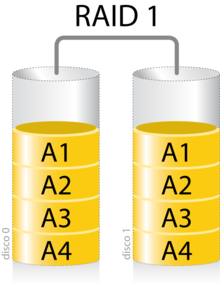
**Nota:** este nivel no pertenece a la familia de los RAIDs ya que justamente no usa redundancia.

**RAID 1**

Se usan **discos espejos**, o sea, la información de cada disco se mantiene siempre duplicada en otro idéntico. O sea, el MTTF (tiempo medio entre fallos, promedio de tiempo entre fallos de un sistema) aumenta notoriamente, pero duplicando el costo.

La redundancia se obtiene directamente duplicando toda la información, de manera que cada disco físico tiene hecha una imagen en otro disco físico. Con este sistema:

* Una petición de lectura será atendida por el disco que le tome menor tiempo realizarla de los dos que contienen la información.
* Cuando se escribe, se debe hacer sobre los dos discos. El tiempo de escritura es el del disco que más tarde de los dos.
* Es fácil recobrarse de la falla de un disco (queda el otro).
* Es seguro y eficiente (especialmente para la lectura), pero es muy caro.

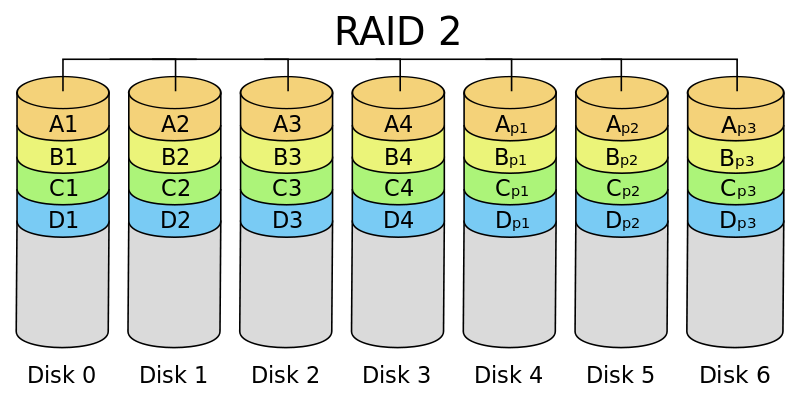


**RAID 2**

Se reduce la redundancia (repetición de la información) usando técnicas de detección y corrección de errores (códigos de Hamming). En vez de hacer un data striping de bloques, hace striping de los bits (para lo que usa strips pequeños de 1 Byte o 1 palabra). Además, sincroniza las cabezas de los discos para que estén todas en la misma posición (es necesario debido a su método de detección de errores).

Se utilizan dos grupos de discos. En un grupo de discos se guardan los datos, en el otro grupo se guardan los códigos de corrección de Hamming.

Aunque requiere menos discos que el RAID 1, todavía es muy costoso y no tiene el beneficio del alto desempeño, solo tiene a favor su alta confiabilidad. No es implementado comercialmente.



**RAID 3**

Está organizado como el de nivel dos (los discos también giran en sincronía), pero en vez de usar el código de Hamming, usa un bit de paridad (XOR), por lo que solo requiere un disco de redundancia.

Si un disco falla, su información puede ser reconstruida con los restantes. El punto es que, gracias a que cada controlador usa sumas de chequeo (y suponiendo que además podemos saber cuándo un controlador falla) sí podemos saber qué trozo de la información está errónea. Y sabiendo eso, basta con usar sólo un disco adicional para guardar información de paridad con la cual es posible reconstituir la información original.

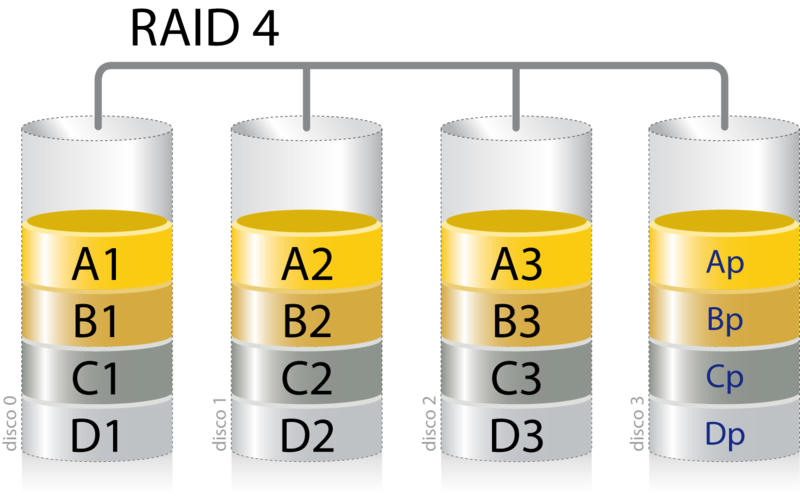
Tiene un buen desempeño para peticiones largas, pero muy pobre para peticiones de archivos pequeños. Además, las escrituras son más lentas que las lecturas, porque al momento de escribir se debe calcular el checksum del archivo.



**RAID 4**

También conocido como IDA (acceso independiente con discos dedicados a la paridad), usa división a nivel de bloques con un disco de [paridad](https://es.wikipedia.org/wiki/Paridad) dedicado. Necesita un mínimo de 3 discos físicos. El RAID 4 es parecido al RAID 3 excepto porque divide a nivel de bloques en lugar de a nivel de [bytes](https://es.wikipedia.org/wiki/Byte). Esto permite que cada miembro del conjunto funcione independientemente cuando se solicita un único bloque. Si la controladora de disco lo permite, un conjunto RAID 4 puede servir varias peticiones de lectura simultáneamente. En principio también sería posible servir varias peticiones de escritura simultáneamente, pero al estar toda la información de paridad en un solo disco, éste se convertiría en el cuello de botella del conjunto.

En el gráfico de ejemplo, una petición del bloque «A1» sería servida por el disco 0. Una petición simultánea del bloque «B1» tendría que esperar, pero una petición de «B2» podría atenderse concurrentemente.



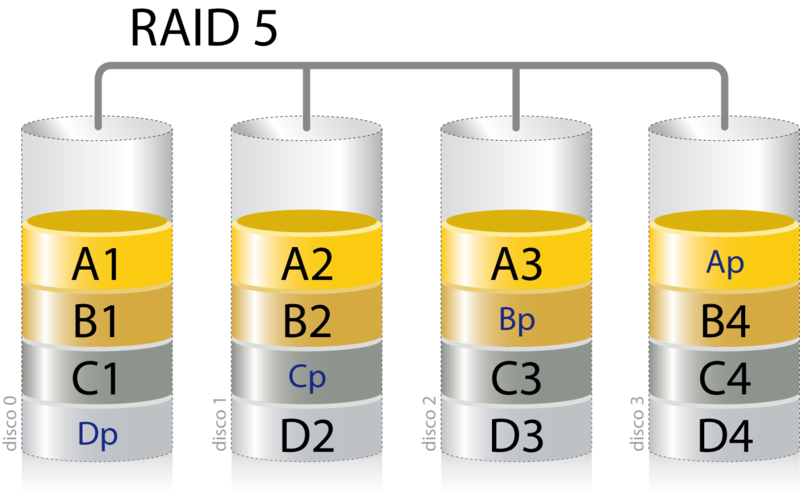
Tanto el nivel 4 como el 5 hacen uso de técnicas de acceso independientes. Aquí cada miembro del disco opera independientemente donde los pedidos de E/S separados pueden ser satisfechos en paralelo. Este RAID más específicamente envuelve una penalidad de escritura cuando un pedido de escritura de un E/S es ejecutado. Cada vez que ocurre una escritura el software del administrador de array debe actualizar no solo el dato del usuario, sino que también la paridad de los bits.

**RAID 5**

Esta organizado en una forma similar al RAID 4. La diferencia radica en que este RAID distribuye sus strips de paridades en todos los discos en vez de temer uno dedicado. Una típica asignación es un esquema de round-robin. Esta distribución a través de todas las unidades evita que se convierte en un cuello de botella para las escrituras en el RAID 4.

Dentro de todo, las escrituras de este nivel suelen ser lentas, pero las lecturas son rápidas y confiables. Además, este tipo de RAID puede soportar la falla de un disco entero y seguir funcionando sin pérdida de datos, ya que la información es reconstruida en base a los otros.

La paridad siempre se guarda en un disco diferente al de los bloques de datos. Es decir, no puedo guardar la paridad del bloque A, en el mismo disco que tengo el bloque A1.

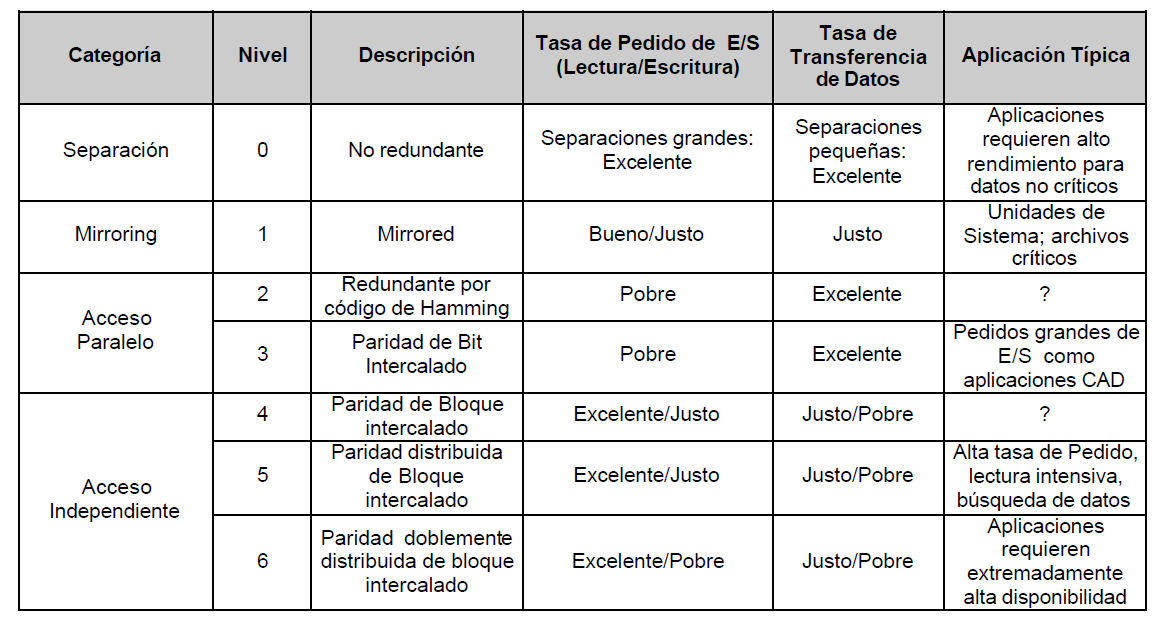


**RAID 6**

Un RAID 6 amplía el nivel RAID 5 añadiendo otro bloque de paridad, por lo que divide los datos a nivel de bloques y distribuye los dos bloques de paridad entre todos los miembros del conjunto. El RAID 6 no era uno de los niveles RAID originales.

El RAID 6 es ineficiente cuando se usa un pequeño número de discos, pero a medida que el conjunto crece y se dispone de más discos la pérdida en capacidad de almacenamiento se hace menos importante, creciendo al mismo tiempo la probabilidad de que dos discos fallen simultáneamente. El RAID 6 proporciona protección contra fallos dobles de discos y contra fallos cuando se está reconstruyendo un disco. En caso de que sólo tengamos un conjunto puede ser más adecuado que usar un RAID 5 con un disco de reserva (hot spare).

Un RAID 6 no penaliza el rendimiento de las operaciones de lectura, pero sí el de las de escritura debido al proceso que exigen los cálculos adicionales de paridad. Esta penalización puede minimizarse agrupando las escrituras en el menor número posible de divisiones (stripes).



**RAIDS Anidados**

Muchas controladoras permiten anidar niveles RAID, es decir, que un RAID pueda usarse como elemento básico de otro en lugar de discos físicos. Resulta instructivo pensar en estos conjuntos como capas dispuestas unas sobre otras.

Al anidar niveles RAID, se suele combinar un nivel RAID que proporcione redundancia con un RAID 0 que aumenta el rendimiento. Con estas configuraciones es preferible tener el RAID 0 como nivel más alto y los conjuntos redundantes debajo, porque así será necesario reconstruir menos discos cuando uno falle. (Así, el RAID 10 es preferible al RAID 0+1 aunque las ventajas administrativas de «dividir el espejo» del RAID 1 se perderían.)

Los niveles RAID anidados más comúnmente usados son:

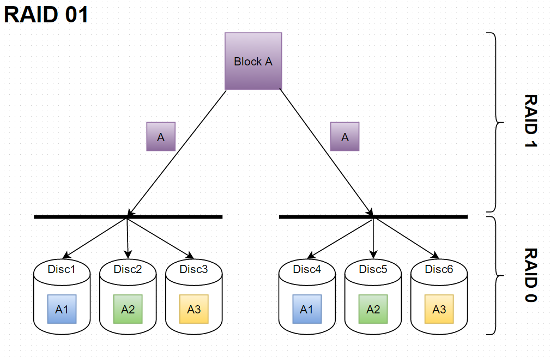
* **RAID 0+1:** Un espejo de divisiones
* **RAID 1+0:** Una división de espejos
* **RAID 30:** Una división de niveles RAID con paridad dedicada
* **RAID 100:** Una división de una división de espejos
* **RAID 10+1:** Un Espejo de espejos

La anidación de niveles anidados se lee de izquierda a derecha. Es decir, el nivel 10 significa que en la capa superior hay un RAID 0, y en la capa inferior un RAID 1.

**RAID 10 VS RAID 01**

Usar RAID 01 es algo estúpido. Siempre se debe utilizar RAID 10.

En RAID 01, lo que hacemos primero es espejar el dato que queremos guardar, y después le aplicamos data striping. Suena bien, pero, basta con que falle solo un disco de uno de los grupos de discos, y perdemos TODA la redundancia.



Si el Disco 2 falla, ya no tenés más redundancia para el bloque A2.

En RAID 10, esto no sucede, porque primero dividís el dato, y espejás cada división.

Explicación detallada acá: <https://muetsch.io/why-raid-10-is-better-than-raid-01.html>

**RAID 0+3 (03 ó 53) y 3+0 (30)**

**Técnica utilizada:** Striping a nivel de byte con paridad dedicada combinada con striping a nivel de bloque.

**Descripción:** Este nivel de RAID es combina los conceptos de striping a nivel de byte, paridad y striping a nivel de bloque. El nivel RAID 30 es más común que el RAID 03. El uso de la paridad, del striping de bloques pequeños (bytes) y el striping de bloques grandes hace que su análisis sea muy difícil. En general provee mejor performance que el RAID 3 debido al uso del striping del RAID 0, pero la misma se sitúa más cerca de la performance del RAID 3 que la del RAID 0, especialmente en las escrituras. El RAID 30 provee mejor tolerancia a fallos y mejor performance en la reconstrucción que el RAID 03, pero ambas dependen más del ancho del stripe del array en RAID 3 que el ancho del array en RAID 0.

Para este nivel de RAID daremos algunos ejemplos para algunas características analizadas, para que los conceptos queden más claros.

**RAID 1+5 (15) y RAID 5+1 (51)**

**Nombre con el que se lo conoce:** RAID 1+5 ó 15 y RAID 5+1 ó 51.

**Técnica utilizada:** Mirroring y striping a nivel de bloque con paridad distribuida.

**Descripción:** Éste es el único nivel de RAID que utiliza ambas técnicas de redundancia, el mirroring y la paridad. Se utiliza principalmente para brindar una altísima tolerancia a fallos y disponibilidad de datos. El nivel RAID 15 se forma creando un conjunto que utiliza el striping con paridad distribuida usando varios pares de discos espejados como componentes. Es muy similar al nivel RAID 15, nada más que el striping se realiza con paridad. En cambio el nivel RAID 51 se forma utilizando dos arrays en RAID 5 y luego espejándolos, similar al RAID 01.

Lo que sobresale en este nivel múltiple de RAID es el nivel de tolerancia a fallos. Un nivel RAID 15 de 8 discos puede tolerar el fallo de 3 discos cualesquiera en todo el conjunto, y lo más sorprendente el nivel RAID 51, también con 8 discos puede tolerar también el fallo de 3 discos y hasta un fallo de 5, siempre y cuando al menos uno de los conjuntos en RAID 5 espejados no tenga más de un fallo. El costo de todo esto es la complejidad y la muy baja eficiencia de almacenamiento.

**Funciones de RAID avanzadas**

**Caching**

Mucha de las controladoras RAID tienen una memoria cache que evita que las operaciones de entrada y salidas tengan que ir directamente a los discos. Si en la controladora RAID se agrega una memoria intermedia de tipo cache, muchas de las operaciones de I/O se podrán resolver directamente en memoria (de la controladora) sin tener necesidad de acceso a los discos.

Así mismo es muy empleada la técnica de “write-back caching”, donde la controladora le dice al SO que la operación de escritura finalizó tan pronto la misma es escrita en la memoria cache de la controladora, y luego éste realiza la escritura real en los discos, acelerando notablemente el tiempo de escritura.

Esta última técnica conlleva un gran riesgo que es la pérdida de datos o la discrepancia en los mismos, por ejemplo frente a la falla de energía. Vean que esta técnica “le dice” al SO que un dato fue escrito en los discos cuando realmente todavía no lo fue. Por lo tanto el SO continúa su ejecución “pensando” que el dato está escrito en memoria secundaria. Si en ese instante la energía de corta del sistema, ese dato nunca va a poder recuperarse. Es por esto que la mayoría de las controladoras del mercado que permiten el uso de esta técnica poseen una batería que permite mantener los datos en la memoria de la controladora hasta que la energía se restablezca y los datos puedan ser volcados a los discos.

Esta técnica presenta las mayores mejoras en sistemas RAID donde la escritura es penalizada, por ejemplo en RAID5 por la obligación de tener que escribir en todos los discos cada vez que se realiza una escritura.

**Swapping**

El swapping no es más que el reemplazo de un disco fallado por uno sano. Dependiendo de cuándo y bajo qué condiciones se puede realizar ese reemplazo, se definen tres tipos de swapping:

* **Hot Swap:** El verdadero hot swap es aquel que permite que el disco sea reemplazado mientras que todo el sistema permanece activo y en línea, es decir, sin interrupción alguno y dando el servicio como si no hubiera ningún fallo (excepto claro está, que el sistema ya está trabajando en modo degradado, pero funcionando y en servicio)
* **Warm Swap:** El warm swap permite realizar el reemplaza mientras que el sistema está encendido, pero con la necesidad de parar toda actividad en los buses de los dispositivos a los que está conectado el dispositivo. Claramente esto hace que el sistema no pueda brindar servicio, pero permanece encendido. Por lo tanto, es peor que el hot swap pero mejor que el cold swap que veremos a continuación.
* **Cold Swap:** En este caso, para efectuar el reemplazo es necesario apagar todo el sistema.

**Hot Spares**

Como ya hablamos, la funcionalidad de hot spare permite dejar uno o más discos como *spares* o repuesto. Estos discos están conectados al sistema pero en modo stand by, es decir, sin usar. Cuando cualquier disco de cualquier array falla (excepto en el RAID 0 que no brinda redundancia), el sistema desconecta de ese array el disco fallado y lo reemplaza automáticamente por el que estaba en modo hot spare. Esto hace que automáticamente el sistema para ese array entre en modo de reconstrucción.

**Array Expansion**

Esta funcionalidad lo que permite es que se puedan expandir con el sistema en servicio el tamaño de los arrays. La manera de hacer esto es agregar nuevos discos al sistema físicamente y luego expandir el tamaño del array en cuestión lógicamente. La mayoría de las controladoras de hoy en día permite esta operatoria.