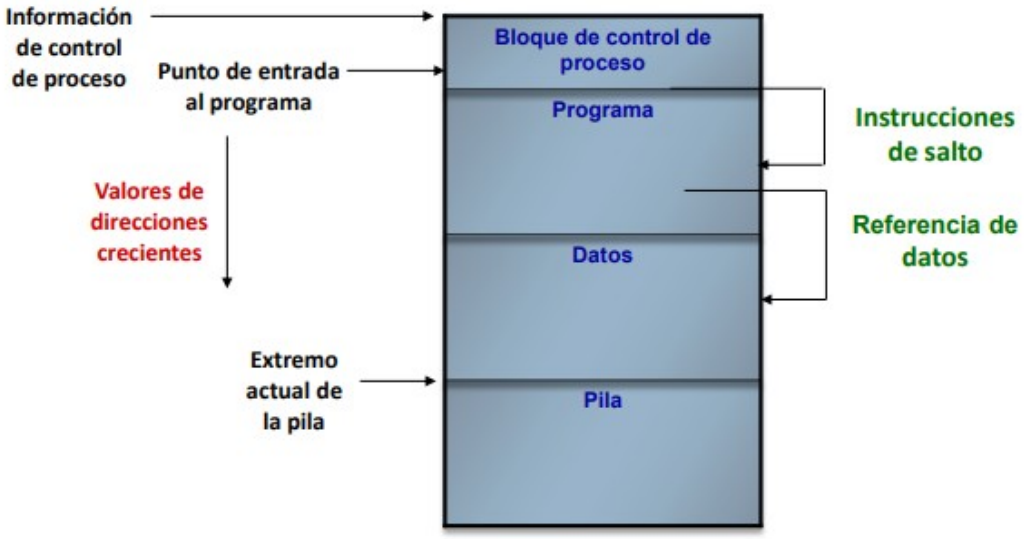
Resumen coloquio sistemas operativos – 2017
Administración de Memoria – Módulo 8
Base
La memoria está compuesta de una gran matriz de palabras, cada una con su propia dirección.
La memoria principal y los registros integrados dentro del propio procesador son las únicas áreas de almacenamiento a las que la CPU puede acceder directamente. Hay instrucciones de máquina que toman como argumentos direcciones de memoria, pero no existe ninguna instrucción que acepte direcciones de disco. Por tanto, todas las instrucciones en ejecución y los datos utilizados por esas instrucciones deberán encontrarse almacenados en uno de esos dispositivos de almacenamiento directo. Si los datos no se encuentran en memoria, deberán llevarse hasta allí antes de que la CPU pueda operar con ellos.
Generalmente, puede accederse a los registros integrados en la CPU en un único ciclo de reloj del procesador. El acceso a memoria puede requerir muchos ciclos del reloj para poderse completar, en cuyo caso el procesador necesitara normalmente detenerse, ya que no dispondrá de los datos requeridos para completar la instrucción que esté ejecutando. Esta situación es intolerable, el remedio consiste en añadir una memoria rápida entre el CPU y la memoria principal, tal memoria rápida se la conoce como memoria cache.
No solo debe preocuparnos la velocidad relativa del acceso a la memoria física, sino que también debemos garantizar una correcta operación que proteja al sistema operativo de los posibles accesos por parte de los procesos de los usuarios y que también proteja a unos procesos de usuario de otros.

Gestión de la memoria
- Subdividir la memoria para acomodar múltiples procesos
- Es necesario asignar la memoria para asegurar una cantidad razonable de procesos listos que consuman el tiempo de procesador disponible.

Requisitos de la gestión de la memoria
- *Reubicación*: El programador no sabe en qué parte de la memoria principal se situará el programa cuando se ejecute. Mientras el programa se está ejecutando, éste puede llevarse al disco y traerse de nuevo a la memoria principal en un área diferente, en tal caso se habla de reubicación. Además deben traducirse las referencias de memoria encontradas en el código del programa en direcciones de memoria físicas.
- *Protección:* Para asegurar una correcta operación se requiere cierta protección de memoria. Primero debemos asegurarnos de que cada proceso disponga de un espacio de memoria separado. Para hacer esto, debemos poder determinar el rango de direcciones legales a las que el proceso pueda acceder y garantizar también que el proceso sólo acceda a esas direcciones legales. Podemos proporcionar esta protección utilizando dos registros, usualmente una base y un límite. El registro base almacena la dirección de memoria física legal más pequeña, mientras que el registro limite específica el tamaño del rango.
  Es imposible comprobar las direcciones absolutas en tiempo de compilación, por lo que, deben comprobarse en el tiempo de ejecución. Es el procesador, en lugar del sistema operativo, el que debe satisfacer el requisito de protección de memoria debido a que el sistema operativo no puede anticipar todas las referencias de memoria que un programa hará.
  Cualquier intento, por parte de un programa que se esté ejecutando en modo usuario, de acceder a la memoria del sistema operativo o a la memoria de los otros usuarios hará que se produzca una interrupción hacia el sistema operativo, que tratará dicho intento como un error

fatal. Este esquema evita que un programa de usuario modifique el código y las estructuras de datos del sistema operativo o de otros usuarios

- *Compartición:* Esta característica permite a varios procesos acceder a la misma porción de memoria principal. Es mejor permitir que cada proceso pueda acceder a la misma copia del programa en lugar de tener su propia copia separada.
- *Organización Física:* La memoria principal disponible para un programa más sus datos podría ser insuficiente. La superposición (overlaying) permite asignar la misma región de memoria a varios módulos. Dada esta técnica el programador no conoce cuánto espacio estará disponible.

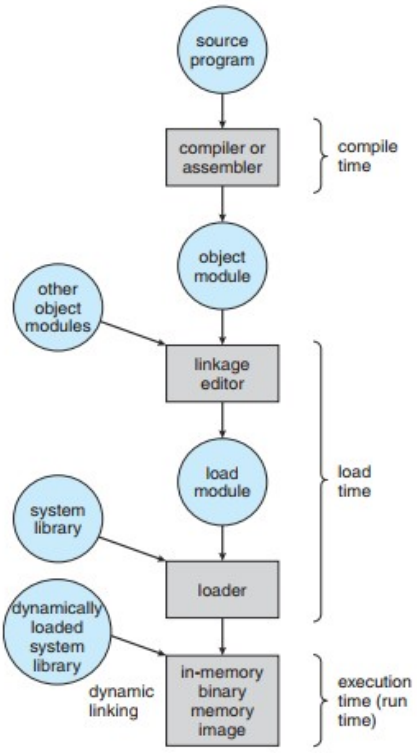


Requisitos de direccionamiento para un proceso

Mapeo de Instrucciones y Datos a Direcciones de Memoria

Clásicamente, la reasignación de instrucciones y datos a direcciones de memoria puede ocurrir en tres etapas diferentes:

- **Tiempo de Compilación:** Si sabemos en el momento de realizar la compilación dónde va a residir el proceso en memoria, podremos generar código absoluto. SI la ubicación inicial cambiase en algún instante posterior, entonces sería necesario recompilar ese código.
- **Tiempo de Carga**: Si no conocemos en tiempo de compilación dónde va a residir el proceso en memoria, el compilador deberá generar código reubicable. En este caso, se retarda la reasignación final hasta el momento de la carga. Si cambia la dirección inicial, tan sólo es necesario volver a cargar el código de usuario para incorporar el valor modificado.
- **Tiempo de ejecución**: Si el proceso puede desplazarse durante su ejecución desde un segmento de memoria a otro, entonces es necesario retardar la reasignación hasta el instante de ejecución. Para que este esquema pueda funcionar, será preciso disponer de hardware especial para el mapeo de direcciones (p.e., registros base-limite).



Carga dinámica

Con la carga dinámica, una rutina no se carga hasta que se la invoca; todas las rutinas se mantienen en disco en un formato de carga reubicable. La ventaja del mecanismo de carga dinámica es que una rutina no utilizada no se cargará nunca en memoria. Este método resulta particularmente útil cuando se necesitan grandes cantidades de código para gestionar casos que sólo ocurren de manera infrecuente, como por ejemplo rutinas de error. En este caso, aunque el tamaño total del programa puede ser grande, la porción que se utilice (y que por tanto se cargue) puede ser mucho más pequeña.

El mecanismo de carga dinámica no requiere ningún soporte especial por parte del sistema operativo. Es responsabilidad de los usuarios diseñar programas que puedan aprovechar este método.

Enlace dinámico

El concepto de enlace dinámico es similar al de la carga dinámica, aunque en este caso lo que se pospone hasta el momento de la ejecución es el montaje, en lugar de la carga. Esta funcionalidad suele emplearse con las bibliotecas del sistema, como por ejemplo las bibliotecas de subrutinas del lenguaje.

Con el enlace dinámico, se incluye un stub dentro de la imagen binaria para cada referencia a una rutina de biblioteca. El stub es un pequeño fragmento de código que indica como localizar la rutina adecuada de biblioteca residente en memoria o como cargar la biblioteca si esa rutina no está todavía presente. Cuando se ejecuta el stub, éste comprueba si la rutina necesaria ya se encuentra en memoria; si no es así, el programa carga en memoria la rutina. En cualquier de los casos, el stub se sustituye así mismo por la dirección de la rutina y ejecuta la rutina.

A diferencia de la carga dinámica, el montaje dinámico suele requerir algo de ayuda por parte del sistema operativo. Si los procesos de la memoria están protegidos unos de otros, entonces el sistema operativo será la única entidad que pueda comprobar si la rutina necesaria se encuentra dentro del espacio de memoria de otro proceso y será también la única entidad que pueda permitir a múltiples procesos acceder a las mismas direcciones de memoria.

Espacio de Direcciones Lógico vs. Espacio de Direcciones Físico

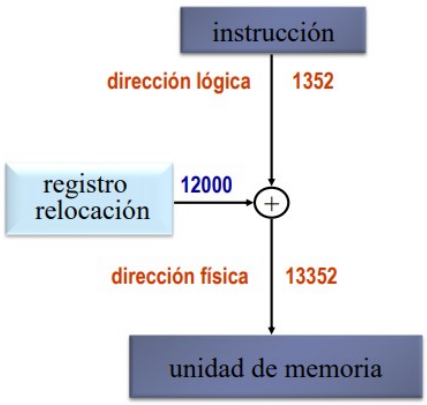
El concepto de espacio de direcciones lógico que está limitado a un espacio de direcciones físicas es central a la administración de la memoria. Una dirección generada por la CPU se denomina dirección lógica (también llamada dirección virtual), mientras que una dirección vista por la unidad de memoria se denomina dirección física.

Las direcciones lógicas y físicas son las mismas en tiempo de compilación y en los esquemas de mapeo de direcciones en tiempo de carga; las direcciones lógicas y físicas difieren en el esquema de mapeo de direcciones en tiempo de ejecución.

Unidad de Administración de Memoria (MMU)

La correspondencia entre direcciones virtuales y físicas en tiempo de ejecución es establecida por un dispositivo de hardware que se denomina unidad de administración de memoria (MMU, memory management unit).

En el esquema MMU, el valor en el registro de locación es agregado a cada dirección generada por el proceso del



usuario en el momento que es presentada a la memoria. Los programas de usuario ven direcciones lógicas, nunca ven direcciones físicas reales.

Intercambio (Swapping)

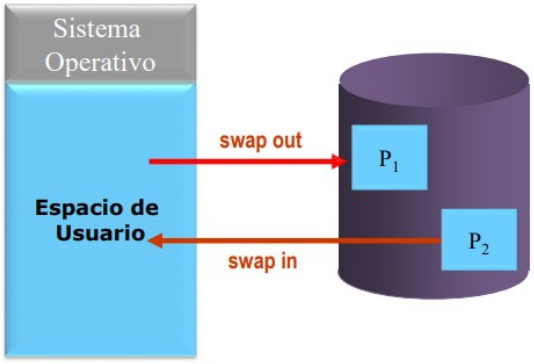
Un proceso debe estar en memoria para ser ejecutado. Sin embargo, los procesos pueden ser intercambiados temporalmente, sacándolos de la memoria y almacenándolos en un almacenamiento de respaldo (backing store), y luego ser vuelto a la misma para continuar su ejecución. Esta estrategia se emplea por ejemplo en un entorno de multiprogramación con el algoritmo de planificación de CPU round robin.

Los mecanismos de intercambio requieren un **backing store**, que normalmente será un disco suficientemente rápido. El disco debe ser también lo suficientemente grande como para poder albergar copias de todas las imágenes de memoria para todos los usuarios, y debe proporcionar un acceso directo a estas imágenes de memoria.

Existe una metodología denominada **roll out, roll in** la cual es una variante del intercambio usado en algoritmo de planificaciones por prioridades; procesos con baja prioridad son intercambiados con procesos de alta prioridad que pueden ser cargados y ejecutados.

La mayor parte del tiempo de intercambio es tiempo de transferencia. El tiempo de transferencia total es directamente proporcional a la cantidad de memoria intercambiada. Además el intercambio está restringido también por otros factores. Si queremos intercambiar un proceso, debemos asegurarnos que esté completamente inactivo.

Actualmente, estos mecanismos estándar de intercambio se utilizan en muy pocos sistemas. Dicho mecanismo requiere de un tiempo muy alto y proporciona un tiempo de ejecución demasiado pequeño como para constituir una solución razonable de gestión de memoria. Sin embargo, lo que si podemos encontrar en muchos sistemas son versiones modificadas de este mecanismo de intercambio.



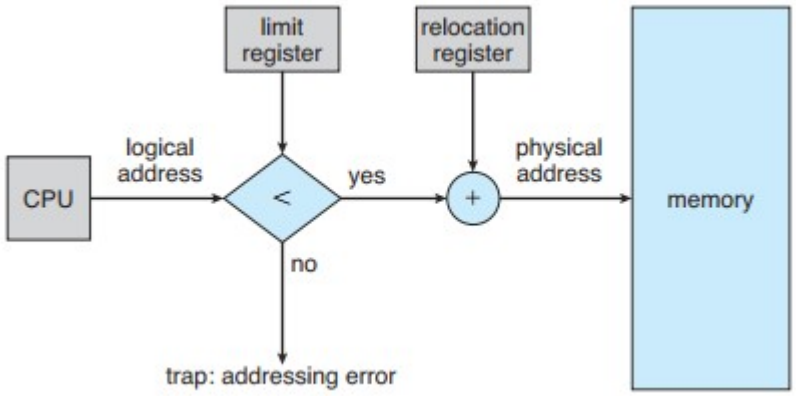
Vista esquemática del intercambio

Alocación Contigua

La memoria principal está usualmente dividida en dos particiones: una para el sistema operativo residente y otra para los procesos de usuario. Podemos situar el sistema operativo en la zona baja o en la zona alta de la memoria. El principal factor que afecta a esta decisión es la ubicación del vector de interrupciones. Puesto que el vector de interrupciones se encuentra a menudo en la parte baja de la memoria, los programadores tienen a situar también el sistema operativo en dicha zona.

Bajo la alocación en partición simple, se usa un esquema de registro de realocación para proteger los procesos de usuario uno de otro, y cambios en el código y datos del SO. El registro

de realocación contiene el valor de la dirección física más pequeña; el registro limite contiene el rango de las direcciones lógicas, cada dirección lógica debe ser menor que el registro limite; la MMU convertirá la dirección lógica dinámicamente sumándole el valor contenido en el registro de reubicación. Esta dirección es la que se envía a la memoria.



Uno de los métodos más simples para asignar la memoria consiste en dividirla en particiones de tamaño fijo. Cada partición puede contener exactamente un proceso, de modo que el grado de multiprogramación estará limitado por el número de particiones disponibles. En este método de particiones múltiples, cuando una partición está libre, se selecciona un proceso de la cola de entrada y se lo carga en dicha partición. Cuando el proceso termina, la partición pasa a estar disponible para otro proceso. En el esquema de particiones fijas, el sistema operativo mantiene una tabla que indica qué partes de la memoria están disponibles y cuales están ocupadas. Inicialmente, toda la memoria está disponible para los procesos de usuario y se considera como un único bloque de gran tamaño de memoria disponible, al que se denomina agujero. Cuando llega un proceso y necesita memoria, busquemos un agujero lo suficientemente grande como para albergar este proceso. Si lo encontramos, sólo se asigna la memoria justa necesaria, manteniendo el resto de la memoria disponible para satisfacer futuras solicitudes. En cualquier momento determinado, tendremos una lista de tamaños de bloque disponibles y una cola de entrada de procesos.

La asignación dinámica de espacio de almacenamiento, se ocupa de cómo satisfacer una solicitud de tamaño n a partir de una lista de agujeros libres. Hay muchas soluciones a este problema, y las estrategias más comúnmente utilizadas para seleccionar un agujero libre entre el conjunto de agujeros disponibles son primer lugar, próximo lugar, mejor lugar y peor lugar.

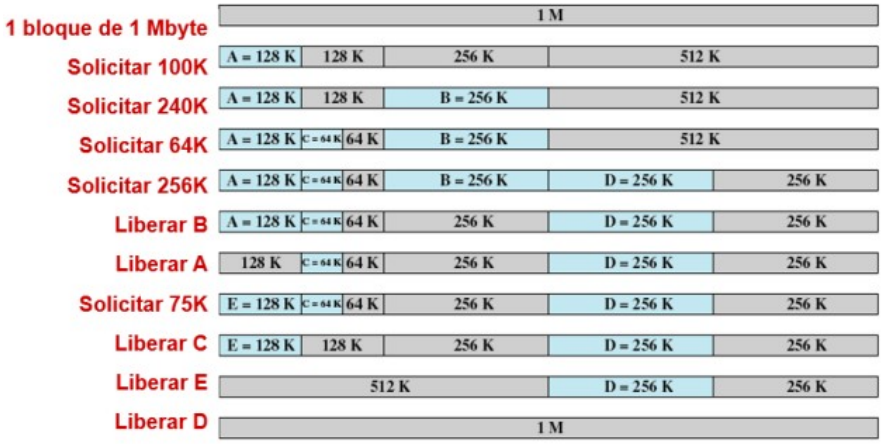
- ⇒ Primer lugar: Aloca en el primer agujero lo suficientemente grande.
- ⇒ Próximo lugar: Aloca en el próximo lugar luego de haber usado el primero.
- ⇒ Mejor lugar: Aloca en el agujero más chico que lo puede alojar; debe buscar en la lista completa, salvo que sea ordenada por tamaño.
- ⇒ Peor lugar: Aloca el agujero más grande; debe buscar en la lista completa. Produce el agujero restante más grande.

Las simulaciones muestran que tanto la estrategia de primer lugar como la de mejor lugar son mejores que la de peor lugar en términos de velocidad y utilización del almacenamiento.

Buddy System (Alocación)

Es un sistema de alocación de memoria utilizado por algunos sistemas operativos. Aquí el espacio completo disponible es tratado como un bloque de tamaño 2^U.

Bajo este esquema, si surge un requerimiento de tamaño s tal que $2^{U-1} < s \leq 2^{U}$, es alocado el bloque completo. De otra manera el bloque es dividido en dos "compañeros" iguales. El proceso continua hasta que es generado el bloque más pequeño mayor o igual que s.



Fragmentación

El problema de la **fragmentación externa** aparece cuando hay un espacio de memoria total suficiente como para satisfacer una solicitud, pero esos espacios disponibles no son contiguos; el espacio de almacenamiento está fragmentado en un gran número de pequeños agujeros. Dependiendo de la cantidad total de espacio de memoria y del tamaño medio de los procesos, esa fragmentación externa puede ser un problema grave o no.

La técnica general para evitar este problema consiste en descomponer la memoria física en bloques de tamaño fijo y asignar la memoria en unidades basadas en el tamaño del bloque. Con esta técnica, la memoria asignada a un proceso puede ser ligeramente superior a la memoria solicitada. En estas ocasiones, puede aparecer entonces la **fragmentación interna**: la memoria ocupada puede ser ligeramente más grande que la demandada.

Otra solución al problema de la fragmentación externa consiste en la compactación. El objetivo es mover el contenido de la memoria con el fin de situar toda la memoria libre de manera contigua, para formar un único bloque de gran tamaño. Sin embargo, la compactación no es siempre posible. La compactación solo es posible si la reubicación es dinámica y se lleva a cabo en tiempo de ejecución. Además pueden surgir problemas de E/S, por lo que se aconseja dejar el trabajo en memoria mientras está involucrando en una E/S y hacer E/S solo en los buffers del SO. El algoritmo de compactación más simple consiste en mover todos los procesos hacia uno de los extremos de la memoria; de esta forma, todos los agujeros se moverán en la otra dirección, produciendo un único agujero de memoria disponible de gran tamaño. Sin embargo, este esquema puede ser muy caro de implementar.

Otra posible solución al problema de la fragmentación externa consiste en permitir que el espacio de direcciones lógicas de los procesos no sea contiguo, lo que hace que podamos asignar memoria física a un proceso con independencia de dónde esté situada dicha memoria. Hay dos técnicas complementarias que permiten implementar esta solución: la paginación y la segmentación y pueden combinarse.

Paginado

La paginación es un esquema de gestión de memoria que permite que el espacio de direcciones físicas de un proceso no sea contiguo. La paginación evita el considerable problema de encajar los fragmentos de memoria de tamaño variable en el backing store. El método básico para implementar la paginación implica descomponer la memoria física en una serie de bloques de tamaño fijo denominados **cuadros** y descomponer la memoria lógica en bloques del mismo

tamaño denominados **páginas**. Cuando hay que ejecutar un proceso, sus páginas se cargan desde el backing store en los cuadros de memoria disponibles. El backing store está dividido en bloques de tamaño fijo que tienen el mismo tamaño que los cuadros de memoria. Se guarda información de todos los cuadros libres y se implementa **una tabla de páginas** que permite traducir las direcciones lógicas a físicas. Para correr un programa de n páginas, se necesita encontrar n cuadros libres y cargar el programa.

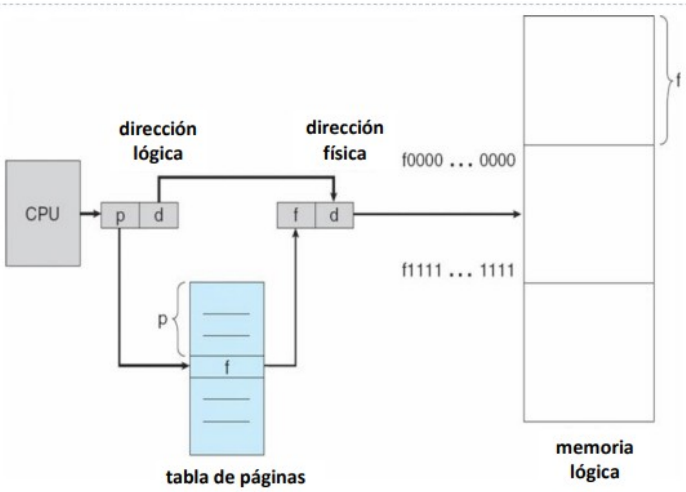
Toda dirección generada por la CPU está dividida en dos partes:

1. Número de página (p): se utiliza como índice para una tabla de páginas. La tabla de páginas contiene la dirección base de cada página en memoria física;
2. Desplazamiento de página (d): se combina con el desplazamiento de página para definir la dirección física que se envía la unidad de memoria.

El tamaño de página al igual que el del cuadro, está definido por el hardware. El tamaño de página es, normalmente, una potencia de 2, variando entre 512 bytes y 16 MB por página, dependiendo de la arquitectura de la computadora. La selección de una potencia de 2 como tamaño de página hace que la traducción de una dirección lógica a un número de página y a un desplazamiento de página resulte particularmente fácil. Si el tamaño del espacio de direcciones lógicas es 2^m y el tamaño de página 2^n unidades de direccionamiento, entonces los m-n bits superiores de cada dirección lógica designaran el número de página, mientras que los n bits inferiores indicarán el desplazamiento de página. Por tanto, la dirección lógica tiene la estructura siguiente:

Donde p es un índice de la tabla de páginas y d es el desplazamiento dentro de la página.

Cuando usamos un esquema de paginación, no tenemos fragmentación externa: todos los cuadros libres podrán ser asignados a un proceso que los necesite. Sin embargo, lo que si podemos tener es un cierto grado de fragmentación interna, ya que los cuadros se asignan como unidades y, si los requisitos de memoria de un proceso no coinciden exactamente con las fronteras de página, el último cuadro asignado puede no estar completamente lleno. Cuando llega un proceso al sistema para ejecutarlo, se examina el tamaño expresado en páginas. Cada página del proceso necesitara un cuadro. Por tanto, si el proceso requiere n páginas, deberá haber disponibles al menos n cuadros en memoria.



Hardware de paginado

Implementación de la tabla de páginas

La tabla de páginas se mantiene en memoria principal, utilizándose un **registro base de tabla de páginas (PTBR, page table base register)** para apuntar a la tabla de páginas. Para cambiar las tablas de páginas, sólo hace falta cambiar este único registro, reduciéndose así sustancialmente el tiempo de cambio de contexto. El **registro de longitud de la tabla de páginas (PTLR)** indica el tamaño de la tabla de páginas. En este esquema cada acceso a instrucción/dato requiere dos accesos a memoria. Uno para la tabla de páginas y otro para la instrucción/dato.

El problema del doble acceso puede ser resuelto por el uso de una cache hardware especial de adelantamiento llamado **registro asociativo o translation look-aside buffers (TLBs**). El búfer TLB es una pequeña memoria asociativa de alta velocidad. El búfer TLB se utiliza con las tablas de página de la forma siguiente: el búfer TLB contiene sólo unas cuantas entradas de la tabla de páginas; cuando la CPU genera una dirección lógica, se presenta el número de página al TLB y si se encuentra ese número de página, su número de cuadro correspondiente estará inmediatamente disponible y se utilizara para acceder a la memoria. Si el número de página no se encuentra en el búfer TLB, es necesario realizar una referencia a memoria para consultar la tabla de páginas. Una vez obtenido el número de cuadro, podemos emplearlo para acceder a la memoria. Además, podemos añadir el número de página y el número de marco al TLB, para poder encontrar los correspondientes valores más rápidamente en la siguiente referencia que se realice. Si el TLB ya está lleno, el sistema operativo deberá seleccionar una de las entradas para sustituirla.

página #	cuadro #

Traducción de dirección (p, d)

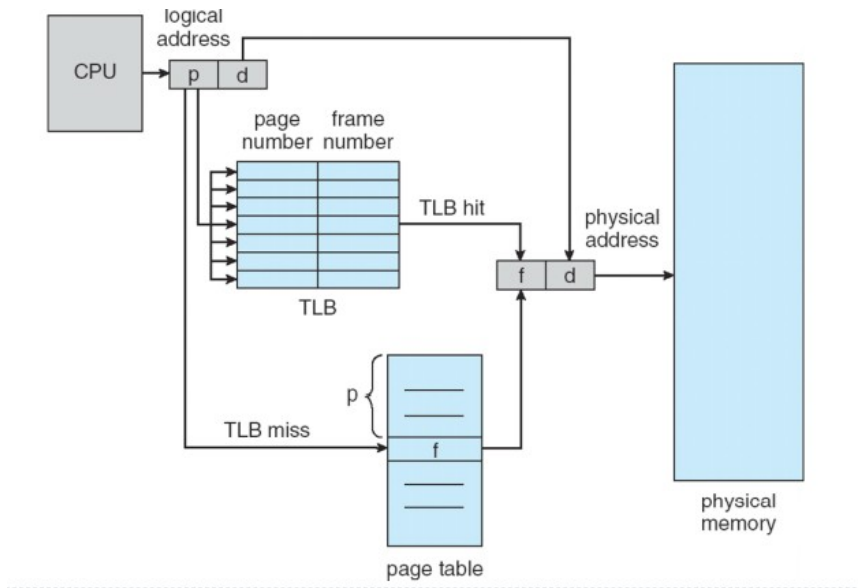
- Si p es un registro asociativo, tome el cuadro #.
- Sino tome el cuadro # desde la tabla de páginas en memoria.

Tiempo Efectivo de Acceso

Supongamos que el ciclo de memoria es de 1 microsegundo y la búsqueda asociativa demanda una unidad de tiempo ε, luego la relación de acierto es el porcentaje de veces que la página es encontrada en los registros asociativos; y está relacionado con el número de registros asociativos.

Si la relación de acierto es α entonces el tiempo efectivo de acceso (TEA) es

$$TEA = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)\,(1 - \alpha) = 2 + \varepsilon - \alpha$$

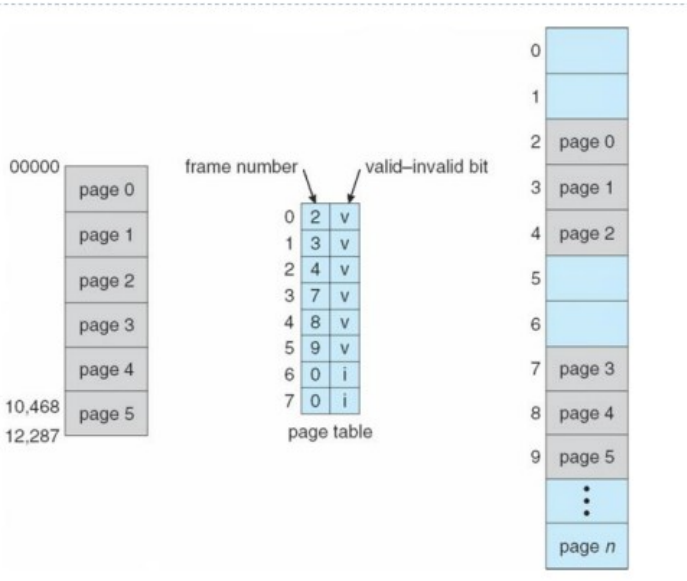


Hardware de paginado con TLB

Protección de Memoria

La protección de memoria en un entorno paginado se consigue mediante una serie de bits de protección asociados con cada cuadro. Normalmente, estos bits se mantienen en la tabla de páginas.

Uno de los bits puede definir una página como de lectura-escritura o de sólo lectura. Podemos ampliar fácilmente este enfoque con el fin de proporcionar un nivel más fino de protección brindando protección de sólo lectura, de lectura escritura o de sólo ejecución, o podemos permitir cualquier combinación de estos accesos, proporcionando bits de protección separados para cada tipo de acceso. Los intentos ilegales provocaran una interrupción hardware hacia el sistema operativo.



Generalmente, se suele asociar un bit adicional con cada entrada de la tabla de páginas, un bit de válido-inválido. Cuando se configura este bit como “valido”, la página asociada se encontrara en el espacio de direcciones lógicas del proceso y será, por tanto, una página legal. Cuando se configura el bit como “inválido”, la página no se encuentra en el espacio de direcciones lógicas del proceso. Las direcciones ilegales se capturan utilizando este bit. El sistema operativo configura este bit para cada página, con el fin de permitir o prohibir el acceso a dicha página.

También se utiliza el registro PTLR como mecanismo de protección. Este se compara con cada dirección lógica para verificar que esa dirección se encuentre dentro del rango de direcciones válidas definidas para el proceso. Si esta comprobación fallara, se produciría una interrupción de error dirigida al sistema operativo.

Tablas de Páginas Multinivel

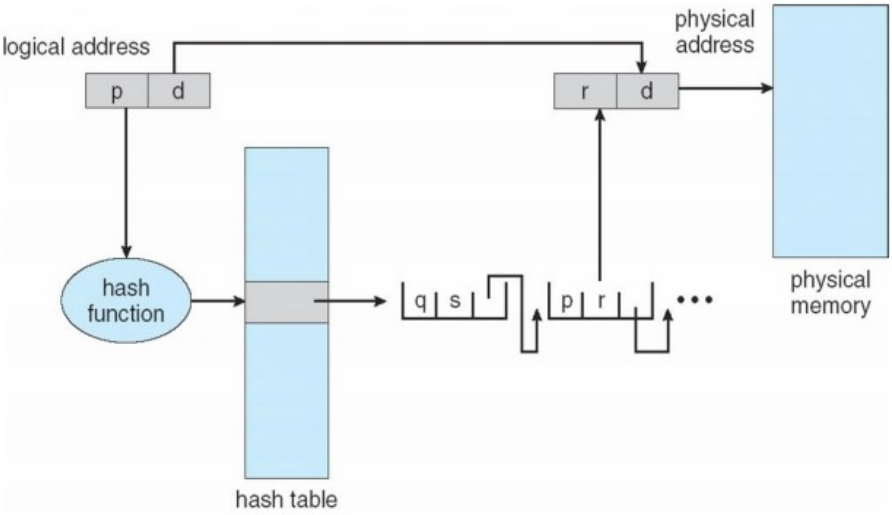
La mayoría de los sistemas informáticos modernos soportan un gran espacio de direcciones lógicas. En este tipo de entorno, la propia tabla de páginas llega a ser excesivamente grande.

Obviamente, no conviene asignar la tabla de páginas de forma contigua en memoria principal. Una solución simple a este problema consiste en dividir la tabla de páginas en fragmentos más pequeños y podemos llevar a cabo esta división en formas distintas.

Una de esas formas consiste en utilizar un algoritmo de paginación en dos niveles, en el que la propia tabla de páginas esta también paginada. Puesto que la traducción de las direcciones se realiza comenzando por la tabla de páginas externa y continuando luego por la interna, este esquema también se conoce con el nombre de tabla de páginas con mapeo directo. También se pueden establecer configuraciones en más niveles.

Tablas de Páginas con Hash

Una técnica común en espacios de direcciones superiores a 32 bits consiste en utilizar una tabla hash de páginas, donde el valor hash es el número de página virtual. Cada entrada de la tabla hash contiene una lista enlazada de elementos que tiene como valor hash una misma ubicación. Cada elemento está compuesto de tres campos: el número de página virtual, el valor del cuadro de página mapeado y un punto del siguiente elemento en la lista enlazada.
El algoritmo funciona de la forma siguiente: al número de página virtual de la dirección virtual se le aplica una función hash para obtener un valor que se utiliza como índice para la tabla hash. El número de página virtual se compara con el campo 1 del primer elemento de la lista enlazada. Si hay una correspondencia, se utiliza el cuadro de página correspondiente para formar la dirección física deseada. Si no se produce una correspondencia, se exploran las subsiguientes entradas de la lista enlazada hasta encontrar el correspondiente número de página virtual.

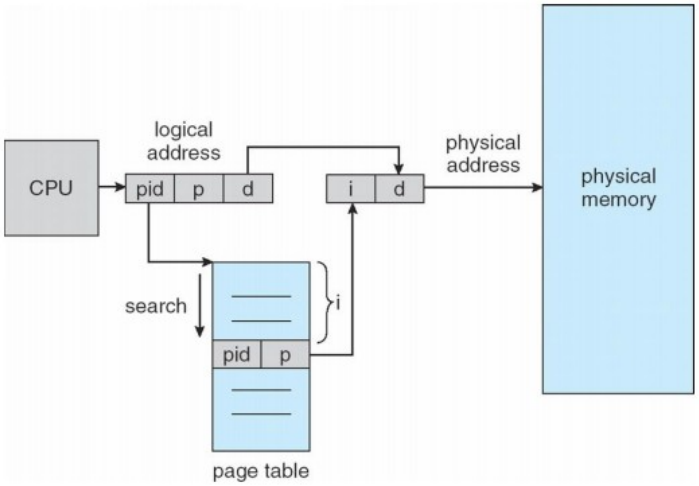


Tablas de páginas con Hash

Tablas de Páginas Invertida

La tabla de páginas incluye una entrada por cada página que el proceso esté utilizando. El sistema operativo debe entonces traducir cada referencia a una dirección física de memoria. Puesto que la tabla esta ordenada según la dirección virtual, el sistema operativo podrá calcular dónde se encuentra en la tabla la entrada de la dirección física asociada y podrá utilizar dicho valor directamente. Pero una de las desventajas de este método es que cada tabla de página puede estar compuesta por millones de entradas. Estas tablas pueden ocupar una gran cantidad de memoria física, simplemente para controlar el modo en que se están utilizando otras partes de la memoria física.
Para resolver este problema, podemos utilizar una tabla de páginas invertida. Las tablas de páginas invertidas tienen una entrada por cada página real o cuadro de la memoria. Cada

entrada está compuesta de la dirección virtual de la página almacenada en dicha ubicación de memoria real, e incluye información acerca del proceso que posee dicha página. Por tanto, en el sistema sólo habrá una única tabla de páginas y esa tabla sólo tendrá una entrada por cada página de la memoria física.
Aunque este esquema permite reducir la cantidad de memoria necesaria para almacenar cada tabla de páginas, incrementa la cantidad de tiempo necesaria para explorar la tabla cuando se produce la referencia a una página. Puesto que la tabla de páginas invertida está ordenada según la dirección física, pero las búsquedas se producen según las direcciones virtuales, puede que sea necesario explorar toda la tabla hasta encontrar una correspondencia y esta exploración consumiría demasiado tiempo. Para aliviar este problema, se utiliza una tabla hash, con el fin de limitar la búsqueda a una sola entrada de la tabla de páginas o a unas pocas entradas.



Arquitectura de la Tabla de Página Invertida

Páginas compartidas

Una ventaja de la paginación es la posibilidad de compartir código común. Esta consideración es particularmente importante en un entorno de tiempo compartido.

⇒ **Código compartido:** Una copia de código de lectura solamente (reentrante) es compartido entre procesos (p.e. editores de texto, compiladores, sistemas de ventanas). El código compartido debe aparecer en la misma locación en el espacio de direcciones de todos los procesos.

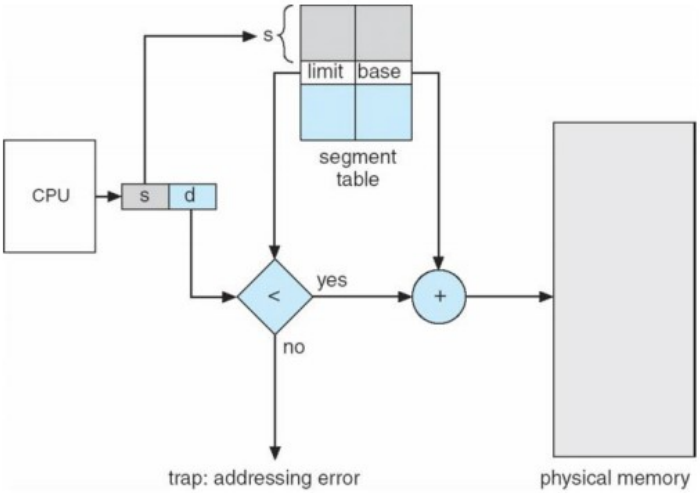
⇒ **Código privado y datos:** Cada proceso guarda una copia separada de esta información y las páginas de código privado y datos pueden aparecer en cualquier lugar del espacio de direcciones lógico.

Segmentación

Los usuarios prefieren ver la memoria como una colección de segmentos de tamaño variable, sin que exista necesariamente ninguna ordenación entre dichos segmentos.
La segmentación es un esquema de gestión de memoria que soporta la visión del usuario de la memoria. Un espacio lógico de direcciones es una colección de segmentos y cada segmento tiene un número de segmento y una longitud. Las direcciones especifican tanto el nombre del segmento como el desplazamiento dentro de ese segmento. El usuario especifica, por lo tanto, cada dirección proporcionando dos valores: nombre de segmento y desplazamiento.
Un programa es una colección de segmentos. Un segmento es una unidad lógica como:

- Programa principal,
- Procedimiento,
- Función,

- Variables globales,
- Variables locales,
- Bloque común,
- Stack,
- Tabla de símbolos,
- Arreglos.



Hardware de segmentación

Arquitectura de Segmentación

Las direcciones especifican tanto el número del segmento como el desplazamiento dentro de este segmento:

<numero-segmento, offset>

La **tabla de segmentos** mapea direcciones físicas en dos dimensiones. Cada entrada de la tabla de segmentos tiene:

- Dirección base del segmento: contiene la dirección física inicial del lugar donde el segmento reside dentro de la memoria
- Límite del segmento: especifica la longitud del segmento.

Además se cuenta con el **registro base de la tabla de segmento (STBR)** que apunta a la locación de la tabla de segmentos en memoria y el **registro de longitud de la tabla de segmentos (STLR**) que indica el número de segmentos usados por el programa, donde el número de segmento s es lega si s<STLR.

Con cada entrada en la tabla de segmentos se asocia un bit de validación. Si el bit de validación es cero entonces el segmento es ilegal. También se puede extender este esquema para brindar una granularidad más fina con privilegios de lectura, escritura o ejecución.

Dado que los segmentos varían en longitud, la alocación de memoria es un problema de alocación dinámica de almacenaje. Para este problema se acude a la utilización de la política de primer lugar y mejor lugar. Claramente esto produce problemas de fragmentación externa. Esto es espacio de memoria de tamaño suficiente para el segmento libre pero no contigua. Por último, cabe subrayar, que para compartir memoria sólo se debe poseer el mismo número de segmento.

Memoria Virtual – Módulo 9

Base

La memoria virtual incluye la separación de la memoria lógica del usuario de la memoria física. Esta separación permite proporcionar a los programadores una memoria virtual extremadamente grande, cuando sólo está disponible una memoria física de menor tamaño.
El espacio de direcciones virtuales de un proceso hace referencia a la forma lógica de almacenar un proceso en la memoria. Típicamente, esta forma consiste en que el proceso comienza en una cierta dirección lógica y está almacenado de forma contigua en la memoria. El espacio de direcciones lógicas puede ser más grande que el espacio de direcciones físicas, y solo parte de un programa necesita estar en memoria para su ejecución.
Cabe destacar que, dejamos que el heap crezca hacia arriba en la memoria, ya que se utiliza para la asignación dinámica de memoria. De forma similar, permitimos que la pila crezca hacia abajo con las sucesivas llamadas a función. El gran espacio vacío entre el heap y la pila forma parte del espacio de direcciones virtual.
Además de separar la memoria lógica de la memoria física, la memoria virtual también permite que dos o más procesos compartan archivos y la memoria mediante mecanismos de compartición de páginas. Esto proporciona las siguientes ventajas:

- ⇒ La memoria virtual reduce la memoria que ocupan los procesos
- ⇒ La memoria virtual puede permitir que se compartan página durante la creación de procesos mediante la llamada al sistema fork(), acelerando así la tarea de creación de procesos.

La memoria virtual puede ser ejecutada vía:

- ❖ Paginado por demanda
- ❖ Segmentación por demanda

Paginado por Demanda

Considerando que generalmente no necesitamos inicialmente todo el programa en la memoria, una alternativa viable consiste en cargar inicialmente las páginas únicamente cuando sean necesarias. Esta técnica se denomina paginación bajo demanda y se utiliza comúnmente en los sistemas de memoria virtual. Con la memoria virtual basada en paginación bajo demanda, sólo se cargan las páginas cuando así se solicita durante la ejecución del programa; de este modo, las páginas a las que nunca se acceda no llegarán a cargarse en la memoria física.
Esta mecánica permite que sean necesarias

- Menos operaciones de E/S,
- Menos memoria física,
- Ayuda a alcanzar una respuesta más rápida,
- Más usuarios.

Se utiliza un intercambiador perezoso el cual jamás intercambia una página con la memoria a menos que esa página vaya a ser necesaria. El intercambiador (swapper) que trata con páginas es un paginador (pager), ya que este último solo se ocupa de páginas individuales de un proceso no de manipular procesos completos como lo hace el intercambiador.

Bit Válido-Inválido

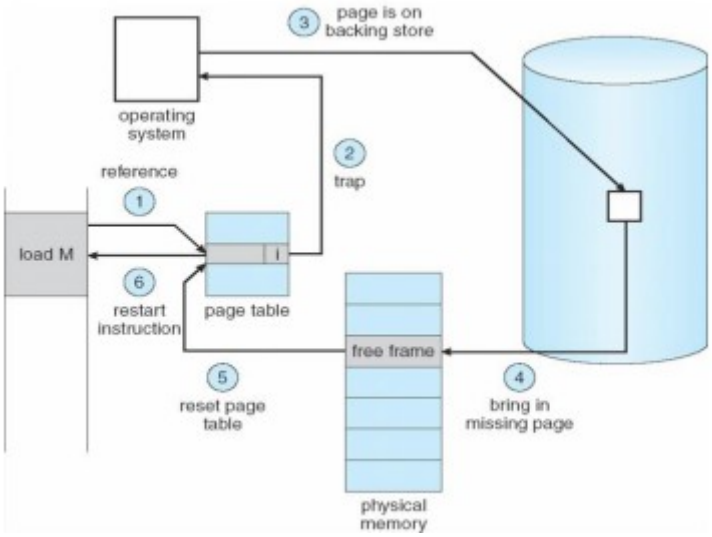
Se necesita algún tipo de soporte hardware para distinguir entre las páginas que se encuentran en memoria y las páginas que residen en disco. Podemos usar para este propósito un bit de válido-inválido el cual se asocia a cada entrada a la tabla de páginas (1 indica “en memoria”, 0 indica “no en memoria”). Inicialmente el bit valido-invalido es puesto en 0 en todas las entradas.

⇒ Cuando se configura este bit como **válido**, la página asociada será legal y además se encontrará en la memoria.
⇒ Si el bit se configura como **inválido**, querrá decir que o bien la página no es válida o es válida pero está actualmente en el disco, por lo que se habla de una falta de página.

Falta de Página

Si el proceso trata de acceder a una página que no haya sido cargada en memoria, se produce una falta de página. El hardware de paginación, al traducir la dirección mediante la tabla de páginas, detectará que el bit de página inválida está activado, provocando una interrupción dirigida al sistema operativo.
El procedimiento para tratar este fallo de página es muy sencillo

1. Comprobamos una tabla interna (que usualmente se mantiene el PCB) correspondiente a este proceso, para determinar si la referencia es un acceso de memoria válido o inválido.
2. Si la referencia era inválida, terminamos el proceso. Si era válida pero esa página todavía no había sido cargada, la cargamos en memoria.
3. Buscamos un cuadro libre (por ejemplo, tomando uno de la lista de cuadros libres).
4. Ordenamos una operación de disco para leer la página deseada en el marco recién llegado.
5. Una vez completada la lectura de disco, modificamos la tabla interna que se mantiene con los datos del proceso y la tabla de páginas para indicar que dicha página se encuentra ahora en memoria.
6. Reiniciamos la instrucción que fue interrumpida. El proceso podrá ahora acceder a la página como si siempre hubiera estado en memoria.



Pasos en el manejo de una Falta de Pagina

Performance del Paginado por Demanda

En el caso extremo, podríamos empezar a ejecutar un proceso sin ninguna página en memoria. Cuando el sistema operativo hiciera que el puntero de instrucciones apuntara a la primera instrucción del proceso, que se encontraría en una página no residente en memoria, se produciría inmediatamente un fallo de página. Después de cargar dicha página en memoria, el proceso continuaría ejecutándose, generando los fallos de página necesarios hasta que todas las páginas requeridas se encontraran en la memoria. A partir de ahí, podría ejecutarse sin ningún

fallo de página. Este esquema sería una **paginación bajo demanda pura**: nunca cargar una página en memoria hasta que sea requerida.
Si bien en teoría, algunos programas podrían acceder a varias nuevas páginas con cada ejecución de una instrucción, posiblemente generando múltiples fallos de página por cada instrucción, y esta situación provocaría una degradación inaceptable en el rendimiento del sistema. Afortunadamente, el análisis de la ejecución de los procesos muestra que este comportamiento es bastante improbable. Los programas tienden a tener lo que se denomina localidad de referencia, por lo que la paginación bajo demanda puede ser una alternativa viable.
El hardware necesario para soportar la paginación bajo demanda es el mismo que para los mecanismos de paginación e intercambio: una tabla de páginas con bits de protección y la memoria secundaria. Un requisito fundamental para la paginación bajo demanda es la necesidad de poder reiniciar cualquier instrucción después de un fallo de página.
La paginación bajo demanda puede afectar significativamente el rendimiento de un sistema informático. Para ver porque vamos a calcular el tiempo efectivo de acceso a una memoria con paginación bajo demanda.
Sea p la probabilidad de que se produzca un fallo de página ($0 \leq p \leq 1$).

- Si p=0 no hay falta de páginas.
- Si p=1 cada referencia es una falta de página.

Cabe esperar que p esté próxima a cero, es decir, que sólo vayan a producirse unos cuantos fallos de página. El tiempo efectivo de acceso será entonces:

TEA = (1-p) x acceso a memoria + p x (sobrecarga de falta de página + salida de la página + entrada de la página + sobrecarga de reinicio).

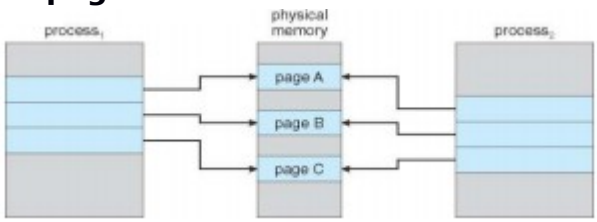
Ejemplo de Paginado por Demanda

Sea el tiempo de acceso a memoria = 200 nanosegundos
Sea el tiempo promedio de servicio de una falta de página = 8 milisegundos.
TEA = (1 - p) x 200 + p x (8 milisegundos) = (1 – p) x 200 + p x 8000000 = 200 + p x 7999800
Podemos ver, entonces, que el tiempo de acceso efectivo es directamente proporcional a la tasa de fallos de página. Si sólo un acceso de cada 1000 provoca un fallo de página, el tiempo efectivo de acceso será de 8,2 microsegundos. La computadora se ralentizará, por tanto, según el factor de 40 debido a la paginación bajo demanda.

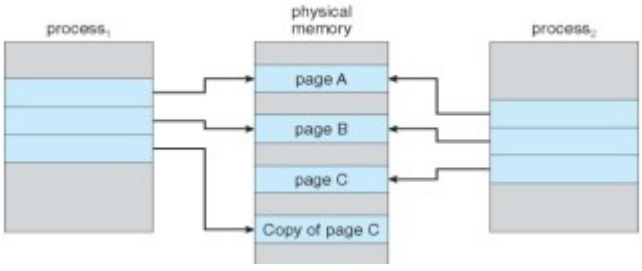
Copia en escritura

Permite que un proceso padre comparta con el proceso hijo (fork()).

⇒ **Antes de modificar la página C**



⇒ **Después de modificar la página C**



¿Qué ocurre cuando no hay cuadros libres?

Imaginemos que, cuando se está ejecutando un proceso de usuario, se produce un fallo de página. El sistema operativo determina dónde reside la página deseada dentro del disco y entonces se encuentra con que no hay ningún cuadro libre en la lista de cuadros libres; toda la memoria está siendo utilizada.

Si bien el sistema operativo dispone de varias posibilidades en este punto, la solución más comúnmente utilizada es el reemplazo de páginas.

Reemplazo de Páginas

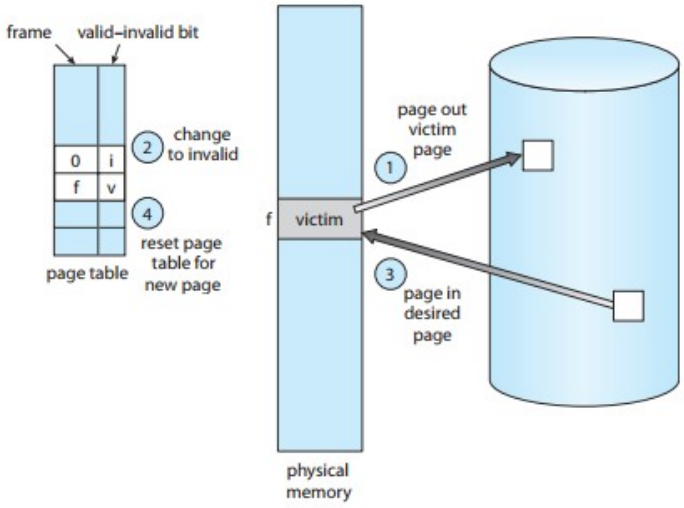
La sustitución de páginas usa la siguiente técnica: Si no hay ningún marco libre, localizamos uno que no esté siendo actualmente utilizado y lo liberamos.

Modifiquemos la rutina de servicio del fallo de página para incluir este mecanismo de sustitución de páginas:

1. Hallar la ubicación de la página deseada dentro del disco.
2. Localizar un marco libre:
   a. Si hay un marco libre, utilizarlo
   b. Si no hay ningún marco libre, utilizar un algoritmo de sustitución de páginas para seleccionar un marco víctima.
   c. Escribir el marco victima en el disco; cambiar las tablas de páginas y de cuadros correspondientes.
3. Leer la página deseada y cargarla en el marco recién liberado; cambiar las tablas de páginas y de cuadros.
4. Reiniciar el proceso de usuario.

Debe observarse que, si no hay ningún marco libre, se necesitan dos transferencias de páginas (una descarga y una carga). Esta situación duplica, en la práctica, el tiempo de servicio del fallo de página e incrementa correspondientemente el tiempo efectivo de acceso. Podemos reducir esta carga de trabajo adicional utilizando un **bit de modificación (o bit de sucio**). Cuando se utiliza este esquema, cada página o marco tiene asociado un bit de modificación. El bit de modificación para una página es activado por el hardware cada vez que se escribe en una palabra, para indicar que la página ha sido modificada. Cuando seleccionamos una página para sustitución, examinamos su bit de modificación. Si el bit está activado, sabremos que la página ha sido modificada desde que se leyó del disco. En este caso, deberemos escribir dicha página en el disco. Sin embargo, si el bit de modificación no es activado, la página no habría sido modificada desde que fue leída y cargada en memoria. Por tanto, si la copia de la página en el disco no ha sido sobrescrita no necesitaremos escribir la página de memoria en el disco, puesto que ya se encuentra allí. Este esquema puede reducir significativamente el tiempo requerido para dar servicio a un fallo de página, ya que reduce a la mitad el tiempo de E/S si la página no ha sido modificada.

El mecanismo de sustitución de páginas resulta fundamental para la paginación bajo demanda. Completa la separación entre la memoria lógica y la memoria física y, con este mecanismo, se puede proporcionar a los programadores una memoria virtual de gran tamaño a partir de una memoria física más pequeña.



Sustitución de páginas

Algoritmos de Reemplazo de Páginas

Debemos resolver dos problemas principales a la hora de implementar paginación bajo demanda: hay que desarrollar un algoritmo de asignación de cuadros y un algoritmo de sustitución de páginas. Si tenemos múltiples procesos en memoria, debemos decidir cuántos cuadros asignar a cada proceso. Además, cuando se necesita una sustitución de páginas, debemos seleccionar los cuadros que hay que sustituir. Debido a que existen muchos algoritmos de sustitución de páginas, en general, intentaremos seleccionar aquel que tenga la tasa de fallos de página más baja.

Podemos evaluar un algoritmo ejecutándolo con una cadena concreta de referencias de memoria y calculando el número de fallos de páginas. La cadena de referencias a memoria se denomina cadena de referencia.

Reemplazo de Páginas Primero en entrar –Primero en salir (FIFO)

El algoritmo más simple de sustitución de páginas es el algoritmo FIFO (first-in, first out). El algoritmo de sustitución FIFO asocia con cada página el instante en que dicha página fue cargada en memoria. Cuando hace falta sustituir una página, se elige la página más antigua. Con objeto de registrar el instante, se utiliza una cola FIFO para almacenar todas las páginas y sustituir la página situada al principio de la cola. Cuando se carga en memoria una página nueva, se la inserta al final de la cola. Este algoritmo de sustitución es fácil de entender y de programar. Sin embargo, su rendimiento no siempre es bueno. Por un lado, la página sustituida puede ser un módulo de inicialización que hubiera sido empleado hace mucho tiempo y que ya no es necesario, también podría contener una variable muy utilizada que fuera inicializada muy pronto y se utilice de forma constante.

Este algoritmo sufre de la anomalía de Belady: para algunos algoritmos de sustitución de páginas la tasa de fallos de página puede incrementarse a medida que se incrementa el número de cuadros asignados.

- Secuencia de referencia: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- 3 cuadros (hay 3 páginas en memoria al mismo tiempo por proceso)

1	1	4	4	4	5	5	5	5	5	5	9 faltas de páginas
2	2	2	1	1	1	1	1	3	3	3	
3	3	3	3	2	2	2	2	2	4	4	

- 4 cuadros

1	1	1	1	5	5	5	5	4	4	10 faltas de páginas
2	2	2	2	2	1	1	1	1	5	
3	3	3	3	3	3	2	2	2	2	
4	4	4	4	4	4	4	3	3	3	

Reemplazo de Páginas Óptimo

Uno de los resultados de la anomalía de Belady fue la búsqueda de un algoritmo óptimo de sustitución de páginas. Un algoritmo óptimo de sustitución de páginas es aquel que tenga la tasa más baja de fallos de página de entre todos los algoritmos y que nunca este sujeto a la anomalía de Belady. A este algoritmo también se lo conoce como mínimo. Consiste simplemente en sustituir la página que no vaya a ser utilizada durante el período de tiempo más largo. La utilización de este algoritmo de sustitución de página garantiza la tasa de fallos de página más baja posible para un número fijo de cuadros.

Desafortunadamente, el algoritmo óptimo de sustitución de páginas resulta difícil de implementar, porque requiere un conocimiento futuro de la cadena de referencia. Como resultado, el algoritmo óptimo se utiliza principalmente con propósitos comparativos.

- Ejemplo con 4 cuadros

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	1	1	1	4	4		6 faltas de páginas
2	2	2	2	2	2	2	2	2		
3	3	3	3	3	3	3	3	3		
4	4	4	5	5	5	5	5	5		

Reemplazo de Páginas LRU

La distinción clave entre los algoritmos FIFO y OPT es que el algoritmo FIFO utiliza el instante en el que se cargó una página en memoria, mientras que el algoritmo OPT utiliza el instante en el que hay que utilizar una página. Si utilizamos el pasado reciente como aproximación del futuro próximo, podemos sustituir la página que no haya sido utilizada durante el periodo más largo de tiempo. Esta técnica se conoce como algoritmo LRU (least-recently-used, menos recientemente usada).

Este algoritmo asocia con cada página el instante correspondiente al último uso de dicha página. Cuando hay que sustituir una página, el algoritmo LRU selecciona la página que no haya sido utilizada durante el periodo de tiempo más largo. Esta política de LRU se utiliza a menudo como algoritmo de sustitución de páginas y se considera que es bastante buena. El principal problema es cómo implementar ese mecanismo de sustitución LRU ya que puede requerir una

considerable asistencia del hardware. El problema consiste en determinar un orden para los cuadros definido por el instante correspondiente al último uso.

Existen dos posibles implementaciones:

- Contadores: En el caso más simple, asociamos con cada entrada en la tabla de páginas un campo de tiempo de uso y añadimos a la CPU un contador. Cuando se realiza una referencia a una página, se copia el contenido del registro de reloj en el campo de tiempo de uso de la entrada de la tabla de páginas correspondiente a dicha página. A pesar de que este esquema nos permite contar siempre con el tiempo de la última referencia a cada página, cuando una página necesita ser reemplazada, se requiere realizar una búsqueda en la tabla de páginas para localizar la página menos recientemente utilizada y realizar una escritura en memoria para cada acceso a memoria.
- Pila: Aquí se mantiene una pila de números de página. Cada vez que se hace referencia a una página, se extrae esa página de la pila y se la coloca en la parte superior. De esta forma, la página más recientemente utilizada se encontrará siempre en la parte superior de la pila y la menos recientemente utilizada en la inferior. Puesto que es necesario eliminar las entradas de la parte intermedia de la pila, lo mejor para implementar este mecanismo es utilizar una lista doblemente enlazada. Si bien cada actualización es más costosa que con otros métodos, no hay necesidad de buscar la página que hay que sustituir.

Tanto el algoritmo óptimo como LRU pertenecen a una clase de algoritmos de sustitución de páginas, denominada algoritmos de pila, que jamás pueden exhibir la anomalía de Belady. Un algoritmo de pila es un algoritmo para el que puede demostrarse que el conjunto de páginas en memoria estaría formado por las n páginas más recientemente referenciadas. Si se incrementa el número de cuadros, estas n páginas seguirán siendo las más recientemente referencias y, por tanto, continuaran estando en la memoria.

Algoritmos de Aproximación a LRU

Pocos sistemas informáticos proporcionan el suficiente soporte hardware como para implementar un verdadero algoritmo LRU de sustitución de páginas. De mínima, algunos sistemas proporcionan un bit de referencia.
Inicialmente, todos los bits son desactivados por el sistema operativo. A medida que se ejecuta un proceso de usuario, el bit asociado con cada una de las páginas referenciadas es activado por el hardware. Después de un cierto tiempo, podemos determinar qué páginas se han utilizado y cuáles no examinado los bit de referencia, aunque no podremos saber el orden de utilización de las páginas. Esta información constituye la base para muchos algoritmos de sustitución de páginas que traten de aproximarse al algoritmo de sustitución LRU.

Algoritmos de Reemplazo de Páginas Segunda Oportunidad (Reloj)

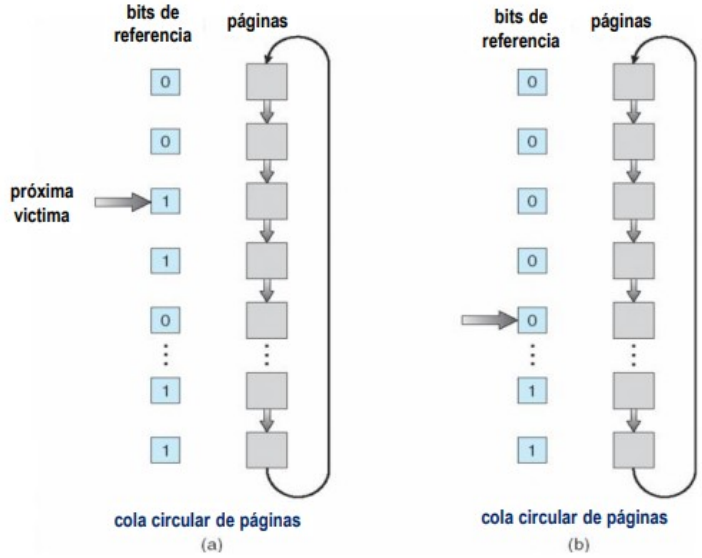
El algoritmo básico de segunda oportunidad para la sustitución de páginas es un algoritmo de sustitución FIFO. Cuando se selecciona una página, inspeccionamos su bit de referencia:

⇒ Si el bit de referencia tiene valor es 0, sustituimos dicha página,
⇒ Si el bit de referencia tiene el valor 1, damos a esa página una segunda oportunidad y seleccionamos la siguiente página FIFO. Cuando una página obtiene una segunda oportunidad, su bit de referencia se borra y el campo indica que su instante de llegada se reconfigura, asignándole el instante actual. Así, si una página se utiliza de forma lo

suficientemente frecuente como para que si bit de referencia permanezca activado, nunca será sustituida.

Una forma de implementar el algoritmo del reloj es utilizando una cola circular. Debemos observar que, en el peor de los casos, cuando todos los bits estén activados, el puntero recorrerá la cola completa, dando a cada una de las páginas una segunda oportunidad y borrando todos los bits de referencia antes de seleccionar la siguiente página que hay que sustituir. En este caso resulta en un algoritmo de sustitución FIFO.

También se pude mejorar este algoritmo considerando el bit de referencia y el bit de modificación. La principal diferencia radica que ahora al poder conocer si una página fue modificada o no, damos preferencia a aquellas páginas que no hayan sido modificadas, con el fin de reducir el número de operaciones de E/S requeridas.



Algoritmo de Reemplazo de Paginas- Segunda oportunidad (Reloj)

Algoritmos de Cuenta

En este se mantiene un contador del número de referencias que se hayan hecho a cada página y puede implementarse mediante los siguientes dos esquemas:

⇒ **El algoritmo de sustitución de páginas LFU (least frequently used, menos frecuentemente utilizada)** requiere sustituir la página que tenga el valor más pequeño del contador. La razón para esta selección es que las páginas más activamente utilizadas deben tener un valor grande en el contador de referencias. Sin embargo, puede surgir un problema cuando una página se utiliza con gran frecuencia durante la fase inicial de un proceso y luego ya no se la vuelve a utilizar, ya que tendrá un valor de contador grande y no será reemplazada. Una solución consiste en desplazar una posición a la derecha los contenidos del contador a intervalos regulares, obteniendo así un contador de uso medio con decrecimiento exponencial.

⇒ **El algoritmo de sustitución de páginas MFU (most frequently used, más frecuentemente utilizada)** se basa en el argumento de que la página que tenga el valor de contador más pequeño acaba probablemente de ser cargada en memoria y todavía tiene que ser utilizada.

Alocación de Cuadros

Si bien está claro que al proceso de usuario se le asignan todos los cuadros libres, la asignación de cuadros está restringida de varias maneras. No podemos, por ejemplo, asignar un número de

cuadros superior al número de cuadros disponibles. Asimismo, debemos asignar al menos un número mínimo de cuadros.
Una razón para asignar al menos un número mínimo de cuadros se refiere al rendimiento. Obviamente, a medida que el número de cuadros asignados a un proceso se reduzca, se incrementará la tasa de fallos de páginas, ralentizando la ejecución del proceso. Además, debemos recordar que, cuando se produce un fallo de página antes de completar la ejecución de una instrucción, dicha instrucción debe reiniciarse. El número mínimo de cuadros está definido por la arquitectura informática y el número máximo está definido por la cantidad de memoria física disponible.
Ejemplo: IBM 370 – 6 páginas para manejar la instrucción SS MOVE:

- ✓ *la instrucción es de 6 bytes, puede expandirse a 2 páginas.*
- ✓ *2 páginas para manejar desde.*
- ✓ *2 páginas para manejar hacía.*

Dos esquemas de alocación:

- ❖ Alocación fija
- ❖ Alocación con prioridad

Alocación Fija

- Alocación igualitaria: La forma más fácil de repartir m cuadros entre n procesos consiste en dar a cada uno un número igual de cuadros, m/n. Por ejemplo, si hay 100 cuadros y 5 procesos, a cada uno se les da 20 páginas.
- Alocación proporcional: Debemos considerar que los diversos procesos necesitaran cantidades diferentes de memoria, por lo que podría suceder que en la alocación igualitaria se desperdicien cuadros. Para resolver este problema, podemos utilizar la alocación proporcional, asignando memoria a cada proceso de acuerdo con el tamaño de este.

Alocación con Prioridad

Debemos observar que, con los mecanismos de asignación igualitaria y proporcional, a los procesos de alta prioridad se los trata igual que a los de baja prioridad. Sin embargo, por su propia definición puede que convenga proporcionar al proceso de alta prioridad más memoria con el fin de acelerar su ejecución, en detrimento de los procesos de baja prioridad. Una solución consiste en utilizar un esquema de asignación proporcional en el que el cociente de cuadros dependa no de tamaño relativo de los procesos, sino más bien de las prioridades de los procesos, o bien de una combinación del tamaño de la prioridad.
Si el proceso Pi genera una falta de página

- ➢ Se selecciona para reemplazar uno de sus cuadros.
- ➢ Se selecciona para reemplazar un cuadro de un proceso con menor número de prioridad.

Reemplazo Global vs. Local

Otro factor importante en la forma de asignar los cuadros a los diversos procesos es el mecanismo de sustitución de página. Si hay múltiples procesos compitiendo por los cuadros, podemos clasificar los algoritmos de sustitución de páginas en dos categorías amplias:

1. Sustitución global: permite a un proceso seleccionar un cuadro de sustitución de entre el conjunto de todos los cuadros, incluso si dicho cuadro está asignado actualmente a algún otro proceso; es decir, un proceso puede quitar un marco a otro.
2. Sustitución local: requiere, por el contrario, que cada proceso sólo efectúe esa selección entre su propio conjunto de cuadros asignado.

Thrashing

Si un proceso no tiene el número de cuadros que necesita para soportar las páginas que están siendo usadas activamente, generará rápidamente fallos de página. En ese momento, deberá sustituir alguna página; sin embargo, como todas sus páginas se están usando activamente, se verá forzado a sustituir una página que va a volver a ser necesaria enseguida. Como consecuencia, vuelve a generar rápidamente una y otra vez sucesivos fallos de página, sustituyendo páginas que se forzado a recargar inmediatamente.

El thrashing provoca graves problemas de rendimiento como se verá a continuación. Cuando el planificador de la CPU ve que la tasa de utilización de la CPU ha descendido, incrementa el grado de multiprogramación. El nuevo proceso tratará de iniciarse quitando cuadros a los procesos que se estuvieran ejecutando, haciendo que se provoquen más fallos de página y que la cola del dispositivo de paginación crezca. Como resultado, la utilización de la CPU cae todavía más y el planificador de la CPU trata de incrementar el grado multiprogramación todavía en mayor medida. Se ha producido thrashing, la tasa de procesamiento desciende vigorosamente, a la vez que se incrementa enormemente la tasa de fallos de página.

Podemos limitar los efectos del thrashing utilizando un algoritmo de sustitución local. Con la sustitución local, si uno de los procesos entra en thrashing, no puede robar cuadros de otro proceso y hacer que este también entre en thrashing. Sin embargo, esto no resuelve el problema completamente. Si los procesos están en thrashing, se pasarán la mayor parte del tiempo en la cola del dispositivo de paginación. De este modo, el tiempo medio de servicio para un fallo de página se incrementará, debido a que ahora el tiempo medio invertido en la cola del dispositivo de paginación es mayor. Por tanto, el tiempo efectivo se incrementará incluso para los procesos que no estén en thrashing.

Método de Conjunto de Trabajo (Working-Set)

Para prevenir el thrashing, debemos proporcionar a los procesos tantos cuadros como necesiten. Para determinar este valor pueden utilizarse varias técnicas distintas. La estrategia basada en el working-set comienza examinando cuántos cuadros está utilizando realmente un proceso, esta técnica define el modelo de localidad de ejecución de un proceso. En este modelo, a medida que un proceso se ejecuta, se va desplazando de una localidad a otra. Una localidad es el conjunto de páginas que se utilizan activamente de forma combinada. Todo programa está generalmente compuesto de varias localidades diferentes, que puede estar solapadas.

Suponga que asignamos a un proceso los suficientes cuadros como para acomodar su localidad actual. El proceso generará fallos de página para todas las páginas de su localidad, hasta que todas ellas estén en memoria; a partir de ahí, no volverá a generar fallos de página hasta que cambie de localidad. Si asignamos menos cuadros que el tamaño de la localidad actual, el proceso entrará en thrashing, ya que no podrá mantener en memoria todas las páginas que esté utilizando activamente.

El modelo del conjunto de trabajo utiliza un parámetro, Δ (número fijo de referencias de páginas), para definir la ventana del conjunto de trabajo. La idea consiste en examinar las Δ referencias de página más recientes. El conjunto de páginas en las Δ referencias de página más recientes es el conjunto de trabajo. Si una página está siendo utilizada, será eliminada del conjunto de trabajo Δ unidades de tiempo después de la última referencia que se hiciera a la misma. Por tanto, el conjunto de trabajo es una aproximación de la localidad del programa.

La precisión del conjunto de trabajo depende de la selección de Δ.

⇒ Si Δ es demasiado pequeña, no abarcara la localidad completa;

⇒ Si Δ es demasiado grande, puede que se solapen varias localidades.

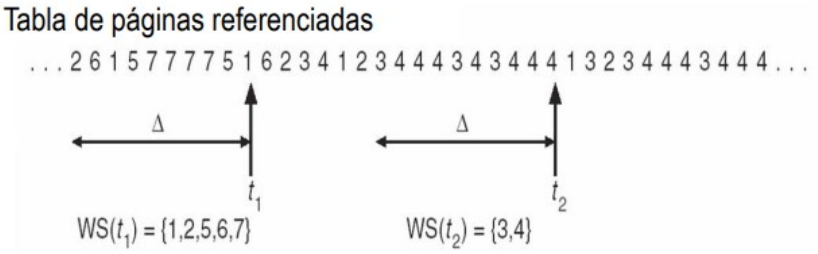
⇒ En el caso extremo, si Δ es infinita, el conjunto de trabajo será el conjunto de páginas utilizadas durante la ejecución del proceso.

La propiedad más importante del conjunto de trabajo es, entonces, su tamaño. Si calculamos el tamaño del conjunto de trabajo, WSS_i, para cada proceso del sistema, podemos considerar que

$$D = \sum WSS_i,$$

Donde D es la demanda total de marcos. Cada proceso estará utilizando activamente las páginas de su conjunto de trabajo. Así, el proceso i necesita WSS_i cuadros. Si la demanda total es superior al número total de marcos disponibles (D > m), se producirá thrashing, porque algunos procesos no dispondrán de suficientes cuadros. En tal caso, el sistema operativo seleccionará un proceso para suspenderlo.

Esta estrategia del working-set impide el thrashing al mismo tiempo que mantiene el grado de multiprogramación con el mayor valor posible. De este modo, se optimiza la tasa de utilización de la CPU.



Modelo de Working-Set

Control Del Working-Set

Aproximar con un intervalo de tiempo + bit de referencia.

Ejemplo: Δ = 10,000

- Las interrupciones del Timer se producen cada 5000 unidades de tiempo.
- Se mantienen en memoria 2 bits por cada página.
- Siempre que el timer interrumpe copia e inicializa los valores de todos los bits de referencia a 0.
- Si uno de los bits en memoria = 1 ⇒ página en el working set.

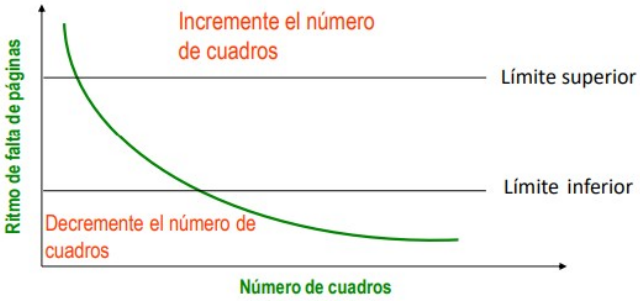
Mejora = 10 bits e interrupción cada 1000 unidades de tiempo.

Esquema de Frecuencia de Falta de Página

Para controlar el thrashing hay una estrategia más directa que está basada en la frecuencia de fallos de página (PFF, page-fault frequency). La idea es establecer un ritmo aceptable de falta de páginas. Si el ritmo actual es demasiado bajo, los procesos pierden cuadros. SI el ritmo actual es demasiado alto, el proceso gana cuadros.

Podemos establecer sendos límites superior e inferior de la tasa deseada de fallos de página.

- Si la tasa real de fallos de página excede el límite superior, eliminamos un cuadro del proceso. Por tanto, podemos medir y controlar directamente la tasa de fallos de página para evitar thrashing.
- Si la tasa de fallos de página se incrementa y no hay ningún cuadro disponible, deberemos seleccionar algún proceso y suspenderlo. Los cuadros liberados se distribuirán entonces entre los procesos que tengan una mayor tasa de fallos de página.



Otras Consideraciones – Prepaginado

Una característica obvia del mecanismo de paginación bajo demanda pura es el gran número de fallos de página que se producen en el momento de iniciar un proceso o reactivarlo. La prepaginación es un intento de evitar este alto nivel de paginación inicial. La estrategia consiste en cargar en memoria de una sola vez todas las páginas que vayan a ser necesarias.

La prepaginación puede ser muy ventajosa en algunos casos. La cuestión es simplemente, si el coste de utilizar la prepaginación es menor que el coste de dar servicio a los correspondientes fallos de página. Podría darse el caso de que muchas de las páginas que fueran cargadas en memoria por el mecanismo de prepaginación no llegaran a utilizarse.

Supongamos que se prepaginan s páginas y que una fragmentación α de esas s páginas llega a utilizarse ($0 \leq \alpha \leq 1$). La cuestión es si el coste de los s x α fallos de página que nos ahorramos es mayor o menos que el coste de prepaginar s x (1 – α) páginas innecesarias. Si α está próxima a 0 la prepaginación no será aconsejable; si α está próxima a 1, la prepaginación será la solución más adecuada.

Otras Consideraciones – Tamaño de Página

Cuando se están diseñando nuevas plataformas, es necesario tomar una decisión en lo referente a cuál es el tamaño de página más adecuado. Como cabría esperar, no hay ningún tamaño de página que sea mejor en términos absolutos. En lugar de ello, lo que hay es una serie de factores que favorecen los distintos tamaños.

Uno de los aspectos que hay que tener en cuenta es el tamaño de la tabla de páginas. Para un espacio de memoria virtual determinado, al reducir el tamaño de página se incrementa el número de páginas y, por tanto, el tamaño de la tabla de páginas. Puesto que un proceso activo debe tener su propia copia de la tabla de páginas, conviene utilizar un tamaño de página grande. Por otro lado, la memoria se aprovecha mejor si las páginas son más pequeñas. Además esto ayuda a reducir la fragmentación interna.

Otro problema es el tiempo requerido para leer o escribir una página. El tiempo de E/S está compuesto de los tiempos de búsqueda, latencia y transferencia. El tiempo de transferencia es proporcional a la cantidad de datos transferida, un hecho que parece actuar a favor de los tamaños de página pequeños. Sin embargo, los tiempos de latencia y búsqueda son mucho mayores que el tiempo de transferencia. Por tanto, el deseo de minimizar el tiempo de E/S es un argumento a favor de los tamaños de página grandes.

Por otro lado, con un tamaño de página pequeño, la cantidad total de E/S debería reducirse, ya que se mejora la localidad. Pero si consideramos un tamaño de página de 1 byte, tendríamos un fallo de página por cada byte. Dado que un fallo de página requiere gran cantidad de trabajo adicional, para minimizar el número de fallos de página debemos disponer de un tamaño de página grande.

Otras Consideraciones – Estructura del Programa

La paginación bajo demanda está diseñada para ser transparente para el programa de usuario. En muchos casos, el usuario no será consciente de la naturaleza paginada de la memoria. En otros casos, sin embargo, puede mejorarse el rendimiento del sistema si el usuario o el compilador conocen el mecanismo subyacente de paginación bajo demanda.

Ejemplo:

- Arreglo de Enteros A[1024,1024]
- Cada fila está almacenada en una página
- Un cuadro

Programa 1

```
for j:=1 to 1024 do
    for i:=1 to 1024
do
    A[i,j]=0;
```

1024 * 1024 faltas de páginas

Programa 2

```
for i:=1 to 1024 do
    for j:=1 to 1024
do
    A[i,j]=0;
```

1024 faltas de páginas

Otras Consideraciones – Fijación de E/S

Cuando se utiliza paginación bajo demanda, en ocasiones es necesario que ciertas páginas queden fijas en memoria. Una de tales situaciones es cuando se realizan operaciones de E/S hacia o desde la memoria de usuario (virtual). La E/S suele implementarse mediante un procesador de E/S independiente.

Debemos asegurarnos de que no se produzca la siguiente secuencia de sucesos: un proceso realiza una solicitud de E/S y es puesto en una cola para dicho dispositivo de E/S; mientras tanto, la CPU se asigna a otros procesos; estos procesos provocan fallos de páginas y uno de ellos, utilizando un algoritmo de sustitución global, sustituye la página que contiene el búfer de memoria del proceso en espera, por lo que dichas páginas se descargan; algún tiempo después, cuando la solicitud de E/S llegue a la parte inicial de la cola del dispositivo, la E/S se realizara utilizando la dirección especificada; sin embargo, dicho cuadro está siendo ahora utilizado para una página distinta, que pertenece a otro proceso.

Existen dos soluciones comunes a este problema: una solución consiste en no ejecutar nunca operaciones de E/S sobre la memoria de usuario. Otra solución consiste en permitir bloquear páginas en memoria. En este caso, se asocia un bit de bloqueo con cada cuadro. Si el cuadro está bloqueado, no podrá ser seleccionado para sustitución.

Sistema de Archivos Interfaz - Módulo 10

Requerimientos esenciales

1. Debe ser posible almacenar gran cantidad de información
2. La información debe sobrevivir a la finalización del proceso que está utilizándola.
3. Múltiples procesos pueden acceder simultáneamente a la información.

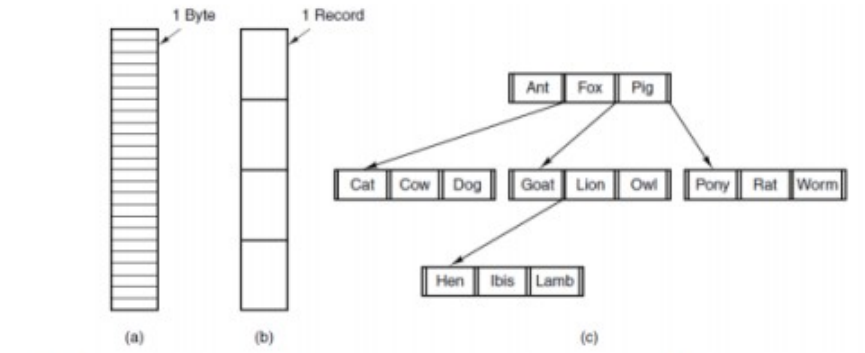
Concepto de Archivo

El sistema operativo realiza una abstracción de las propiedades físicas de los dispositivos de almacenamiento, con el fin de definir una unidad lógica de almacenamiento, el archivo.
Un archivo es una colección de información relacionada, con un nombre, que se graba en almacenamiento secundario. Desde la perspectiva del usuario, un archivo es la unidad más pequeña de almacenamiento secundario lógico.
Comúnmente, los archivos representan programas y datos. Los archivos de datos pueden ser alfabéticos, alfanuméricos o binarios.

Estructura de Archivo

En general, un archivo es una secuencia de bits, bytes, líneas o registros cuyo significado está definido por el creador y el usuario del archivo.
Un archivo tiene una determinada estructura definida que dependerá de su tipo.

- **Ninguna** – secuencia de palabras, bytes.
- **Archivos de estructura simple**, como por ejemplo, un archivo de texto, el cuál es una secuencia de caracteres organizada en líneas (y posiblemente en páginas). Un archivo fuente es una secuencia de subrutinas y funciones, cada una de las cuales está a su vez organizada como una serie de declaraciones, seguidas de instrucciones ejecutables.
- **Archivos de estructura compleja**, como un archivo objeto, que es una secuencia de bytes organizada en bloques que el programa montador del sistema puede comprender.



(a) Secuencia de bytes (b) Secuencia de Registros (c) Árbol

Atributos de Archivo

Los atributos de un archivo varían de un sistema operativo a otro, pero típicamente son los siguientes:

- Nombre: mantiene información en forma legible
- Tipo: necesario para sistemas que soportan diferentes tipo
- Locación: puntero a la locación del archivo en el dispositivo
- Tamaño: tamaño corriente del archivo
- Protección: contrala quien puede leer, escribir y ejecutar el archivo
- Tiempo, fecha, e identificación de usuario: datos para protección, seguridad y visualización de uso.

La información acerca de los archivos se almacena en la estructura de directorios, que también reside en el almacenamiento secundario.

Operaciones sobre Archivos

Un archivo es un tipo de dato abstracto. Para definir un archivo apropiadamente, debemos considerar las operaciones que pueden realizarse con los archivos. El sistema operativo puede proporcionar llamadas al sistema para **crear, escribir, leer, reposicionar, borrar y truncar archivos.**

Además hay dos operaciones importantes

- **Open (F_i):** busca la estructura de directorio en el disco para la entrada F_i, y mueve el contenido de la entrada a la memoria.
- **Close (F_i)**: mueve el contenido de la entrada F_i en la memoria a la estructura del directorio del disco.

Archivos Abiertos

El sistema operativo mantiene una pequeña tabla, denominada tabla de archivos abiertos, que contiene información acerca de todos los archivos abiertos. Cuando se solicita que se realice una operación con un archivo, el archivo se especifica mediante un índice a esta tabla, con lo que no hace falta realizar ninguna exploración de directorio. Cuando el archivo deja de ser utilizado activamente será cerrado por el proceso y el sistema operativo eliminará la entrada correspondiente de la tabla de archivos abiertos.

Son necesarios varios datos para administrar los archivos abiertos:

- Puntero corriente del archivo: El sistema deberá registrar la ubicación correspondiente a la última lecturaescritura, utilizando para ello un puntero de posición actual dentro del archivo. Este puntero es distinto para cada proceso que esté operando con el archivo, y deberá por tanto almacenarse de forma separada de los atributos del archivo almacenados en disco.
- Cuenta de archivo abierto: El contador de aperturas del archivo controla el número de aperturas y cierres y alcanzará el valor cero cuando el último proceso cierre el archivo. Entonces, el sistema podrá eliminar la correspondiente entrada para evitarse quedar sin espacio en la tabla de archivos abiertos.
- Locación en el disco del archivo: La mayoría de las operaciones de archivo requieren que el sistema modifique datos dentro del archivo. La información necesaria para ubicar el archivo en el disco se almacena en la memoria, para que el sistema no tenga que leer de nuevo esa información desde el disco en cada operación.
- Derechos de acceso: Cada proceso abre un determinado archivo con un cierto modo de acceso. Esta información se almacena en la tabla correspondiente a cada proceso, para que el sistema operativo pueda autorizar o denegar las siguientes solicitudes de E/S.

Locking de Archivos Abiertos

Algunos sistemas operativos y sistemas de archivos proporcionan funciones para bloquear un archivo abierto (o determinadas secciones de un archivo). Los bloqueos de archivo resultan útiles para aquellos archivos que son compartidos por varios procesos, como por ejemplo un archivo de registro del sistema que puede ser consultado y modificado por varios procesos distintos del sistema.

Un bloque compartido es similar a un bloqueo lector, en el sentido de que varios procesos pueden adquirir dichos bloqueos concurrentemente. Un bloqueo exclusivo se comparta como un bloqueo escritor; sólo puede adquirir dicho tipo de bloqueo un proceso cada vez.

Modo de implementación

- Mandatorios: Si un bloqueo es mandatorio el acceso es rechazado dependiendo de los locks que se tienen sobre el archivo y los requeridos.
- Flexibles: Alternativamente, si el bloqueo es de tipo flexible los procesos verifican el estado de los locks y deciden que hacer.

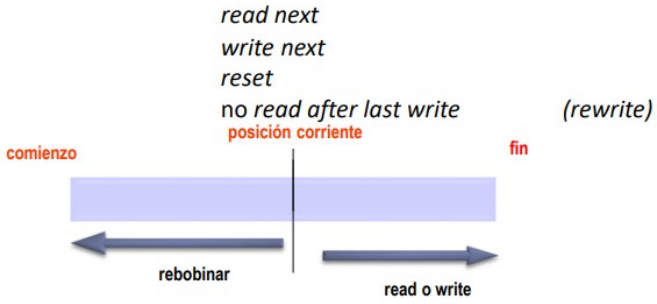
En otras palabras, si el esquema de bloqueos es mandatorio, el sistema operativo garantiza la integridad de los bloqueos, mientras que para el esquema flexible es responsabilidad de los desarrolladores de software garantizar que se adquieran y liberen apropiadamente los distintos bloqueos.

Tipos de Archivo – nombre, extensión

Si un sistema operativo reconoce el tipo de un archivo, podrá operar con ese archivo de formas razonables.
Una técnica común para implementar los tipos de archivo consiste en incluir el tipo como parte del nombre del archivo. El nombre se divide en dos partes: un nombre y una extensión, usualmente separadas por un carácter de punto. De esta forma, el usuario y el sistema operativo pueden determinar, con sólo analizar el nombre, cuál es el tipo de cada archivo.

Métodos de Accesos

Los archivos almacenan información. Cuando hace falta utilizarla, es necesario acceder a esta información y leerla en la memoria de la computadora.
El método de acceso más simple es el **acceso secuencial.** La información del archivo se procesa por orden, un registro después de otro. Las lecturas y escrituras constituyen el grueso de las operaciones realizadas con un archivo. Una operación de lectura (leer siguiente) lee la siguiente porción del archivo e incrementa automáticamente un puntero de archivo, que controla la ubicación de E/S. De forma similar trabaja la operación de escritura. Dichos archivos podrán reinicializarse para situar el puntero al principio de los mismos.



Otro método es el **acceso directo o acceso relativo**. Un archivo está compuesto de registros lógicos de longitud fija que permite a los programas leer y escribir registros rápidamente, sin ningún orden concreto. El método de acceso directo se basa en un modelo de archivos que se corresponde con los dispositivos de disco, ya que los discos permiten el acceso aleatorio a cualquier bloque de un archivo. Para el acceso directo, el archivo se considera como una secuencia numerada de bloques o registros. Así tendremos operaciones tales como leer n, donde n es el número de bloque, en lugar de leer siguiente, y escribir n.
El número de bloque proporcionado por el usuario al sistema operativo es normalmente un número de bloque relativo. Un número de bloque relativo es un índice referido al comienzo del archivo. Así, el primer bloque relativo del archivo será el 0, el siguiente será el 1, aunque cuando las direcciones reales absolutas pueden no ser contiguas. La utilización de números de bloque relativos permite al sistema operativo decidir donde hay que colocar el archivo y ayuda a impedir que el usuario acceda a porciones del sistema de archivos que no formen parte de su archivo.

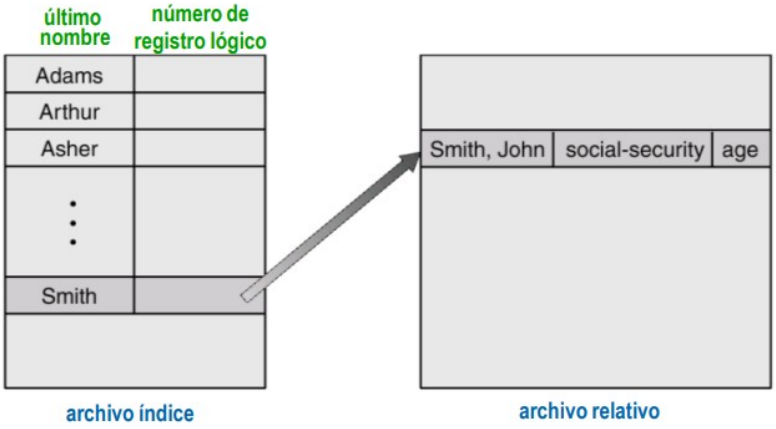
read n
write n
position to n
read next
write next
rewrite n

n= numero relativo del bloque

Ejemplo de Archivo Indexado y Relativo

Pueden construirse otros métodos por encima del método de acceso directo. Estos métodos implican, generalmente, la construcción de un índice para el archivo. El índice, contiene punteros a los distintos bloques. Para encontrar un registro dentro del archivo, primero exploramos el índice y luego usamos el puntero para acceder al archivo directamente y para hallar el registro deseado.

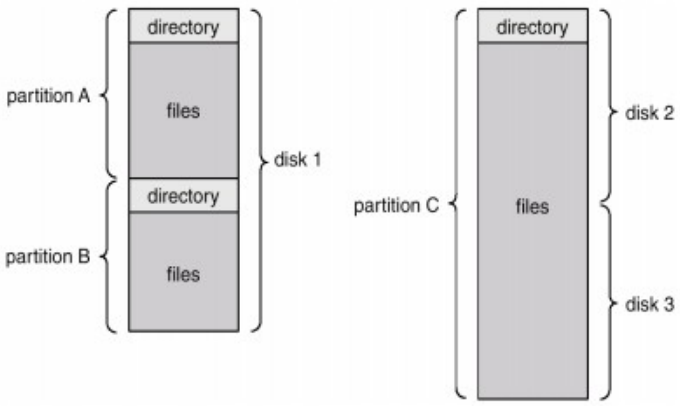
Con los archivos de gran tamaño, el propio archivo índice puede ser demasiado grande como para almacenarlo en memoria. Una solución consiste en crear un índice del archivo índice. El archivo de índice principal contendría punteros a los archivos de índice secundarios que a vez apuntarían a los propios elementos de datos.



Una Organización Típica de un sistema de Archivos

Un disco puede utilizarse completamente para un sistema de archivos. Sin embargo, en ciertas ocasiones es deseable colocar múltiples sistemas de archivos en un mismo disco o utilizar partes de un disco para un sistema de archivos y otras partes para otras cosas, como por ejemplo para espacio de intercambio o como espacio de disco sin formato (raw). Estas partes se conocen como particiones. Las partes también pueden combinarse para formar estructuras de mayor tamaño, conocidas con el nombre de volúmenes.

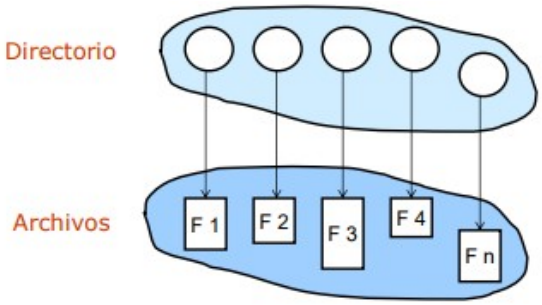
Cada volumen que contenga un sistema de archivos debe también contener información acerca de los archivos almacenados en el sistema. Esta información se almacena como entrada en una tabla de contenidos del volumen. Esta tabla también conocida como directorio almacena información de todos los archivos de dicho volumen, como por ejemplo el nombre, el tipo, la dirección, la longitud



corriente, la máxima longitud, la fecha del último acceso, fecha de la última actualización, el ID del dueño y la información de protección.

Estructura de directorio

Una colección de nodos conteniendo información sobre todo los archivos. La estructura de directorio al igual que los archivos reside en el disco.



Operaciones sobre un Directorio

El directorio puede considerarse como una tabla de símbolos que traduce los nombres de archivo a sus correspondientes entradas de directorio. Si adoptamos esta visión, es fácil comprender que el propio directorio puede organizarse de muchas formas.
Información que puede encontrarse en directorios:

- ⇒ Nombre
- ⇒ Dirección
- ⇒ Longitud corriente
- ⇒ Máxima longitud
- ⇒ Fecha del ultimo acceso
- ⇒ Fecha de la última actualización (para vuelco)
- ⇒ Tipo
- ⇒ ID del dueño
- ⇒ Información de protección

Cuando consideramos una estructura de directorio concreta, tendremos que tener presentes las operaciones que habrá que realizar con el directorio:

- ⇒ Búsqueda de un archivo
- ⇒ Creación de un archivo
- ⇒ Borrado de un archivo
- ⇒ Listado de un directorio
- ⇒ Renombrado de un archivo
- ⇒ Atravesar un sistema de archivos

Organización lógica del directorio

Con la organización lógica de los directorios se busca obtener:

- ❖ Eficiencia: que permita localizar un archivo rápidamente.
- ❖ Nombre: se busca permitir que dos usuarios puedan tener el mismo nombre para diferentes archivos o un mismo archivo puede tener varios nombres diferentes.
- ❖ Agrupamiento: buscan favorecer el agrupamiento lógico de los archivos por propiedades, por ejemplo, todos los programas de Pascal juntos, todos los juegos juntos, etc.

Estructura Arbórea de Directorios

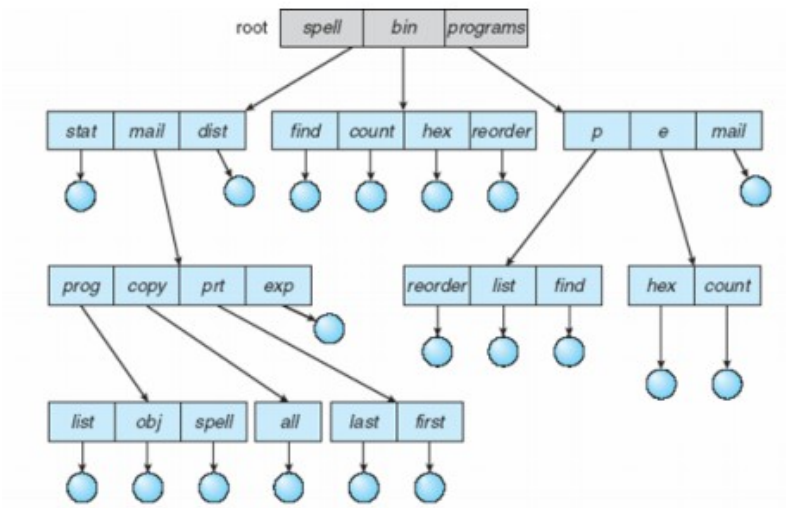
Esta estructura permite a los usuarios crear sus propios subdirectorios y organizar sus archivos correspondientes. Cada árbol tiene un directorio raíz y todos los archivos del sistema tienen un nombre de ruta distinto.

Cada directorio (o subdirectorio) contiene un conjunto de archivos y subdirectorios. Un directorio es simplemente otro archivo, pero que se trata de forma especial. Todos los directorios tienen el mismo formato interno. Un bit de cada entrada de directorio define si esa entrada es un archivo(0) o un subdirectorio(1). Se utilizan llamadas especiales para crear y borrar los directorios.

En el uso normal, cada proceso tiene un directorio actual. Ese directorio actual debe contener la mayor parte de los archivos que actualmente interesen al proceso. Cuando se hace referencia a un archivo que no se encuentre en el directorio actual, entonces el usuario deberá normalmente especificar un nombre de ruta o cambiar el directorio actual, para situarse en el directorio donde fue almacenado ese archivo. Para cambiar de directorio, se proporciona una llamada al sistema que toma como parámetro el nombre del directorio y lo utiliza para redefinir el directorio actual. Los nombres de ruta pueden ser de dos tipos: absolutos y relativos. Un nombre de ruta absoluto comienza en la raíz y sigue una ruta descendiente hasta el archivo especificado, indicando los nombres de los directorios que componen la ruta. Un nombre de ruta relativo define una ruta a partir del directorio actual.

Una decisión interesante de política dentro de un directorio con estructura de árbol es la que se refiere a qué hacer si se borra un directorio. Si el directorio está vacío, podemos simplemente borrar la entrada correspondiente dentro del mismo directorio que lo contuviera. Sin embargo, supongamos que el directorio que hay que borrar no está vacío, sino que contiene varios archivos o subdirectorios. Podemos adoptar una de dos soluciones. Algunos sistemas, como MSDOS, no permiten borrar un directorio a menos que este vacío, esto provoca que el usuario deba realizar una gran cantidad de trabajo. Otra técnica alternativa, como la adoptada por el comando rm de UNIX, consiste en proporcionar una opción: cuando se hace una solicitud para borrar un directorio, también se borran todos los archivos y subdirectorios de dicho directorio. Esta última opción si bien es cómoda, es riesgosa ya que con un solo comando se puede destruir una estructura de directorios completa.

- ✓ Búsqueda eficiente
- ✓ Capacidad de agrupamiento
- ✓ Directorio corriente (directorio de trabajo)
- ✓ Camino de nombres absoluto o relativo



Grafo Acíclico de Directorios

Una estructura de árbol prohíbe la compartición de archivos o directorios. Por el contrario, un grafo acíclico permite que los directorios compartan subdirectorios de archivos. El mismo archivo o subdirectorio puede estar en dos directorios diferentes. El grafo acíclico es una generalización natural del esquema de directorio con estructura de árbol.

Una estructura de directorio en forma de grafo acíclico es más flexible que una estructura simple de árbol, pero también más compleja. Es necesario prestar cuidadosa atención a determinados problemas. Los archivos podrán tener ahora múltiples nombres de ruta absoluta. En consecuencia, puede haber nombres de archivo distintos que hagan referencia a un mismo archivo. La situación es similar al problema de los alias en los lenguajes de programación. Si estamos intentando recorrer el sistema de archivos completo, por ejemplo, para acumular estadísticas, para encontrar un archivo o para copiar todos los archivos en una copia de seguridad; este problema cobra gran importancia, ya que no conviene recorrer las estructuras compartidas más de una vez.

Otro problema es el que se refiere al borrado, ¿cuándo puede desasignarse y reutilizarse el espacio asignado a un archivo compartido? Una posibilidad consiste en eliminar el archivo cuando un usuario cualquiera lo borre, pero esta acción puede dejar punteros colgantes al archivo que ha dejado de existir. El problema puede complicarse si los punteros de archivo restantes contienen direcciones reales de disco y ese espacio se reutiliza subsiguientemente para otros archivos; en ese caso, esos punteros colgantes pueden apuntar a un lugar cualquiera de estos nuevos archivos.

En un sistema en el que la compartición se implemente mediante enlaces simbólicos, la situación es algo más fácil de manejar. El borrado de un enlace no tiene por qué afectar al archivo original, ya que sólo se elimina el enlace. Si lo que se elimina es la propia entrada del archivo, se designará el espacio del archivo, dejando que los enlaces cuelguen. Podemos buscar estos enlaces y eliminarlos también, pero a menos que se mantenga con cada archivo una lista de los enlaces asociados esta búsqueda puede ser bastante costosa. Alternativamente, podemos dejar esos enlaces hasta que se produzca un intento de utilizarlos, en cuyo momento podemos determinar que el archivo del nombre indicado por el enlace no existe y que no podemos resolver el nombre del enlace. Otra técnica de borrado consiste en preservar el archivo hasta que se borren todas las referencias al mismo. Para implementar esta técnica, debemos disponer de algún mecanismo para determinar que se ha borrado la última referencia al archivo. Podríamos mantener un contador de entradas al archivo.

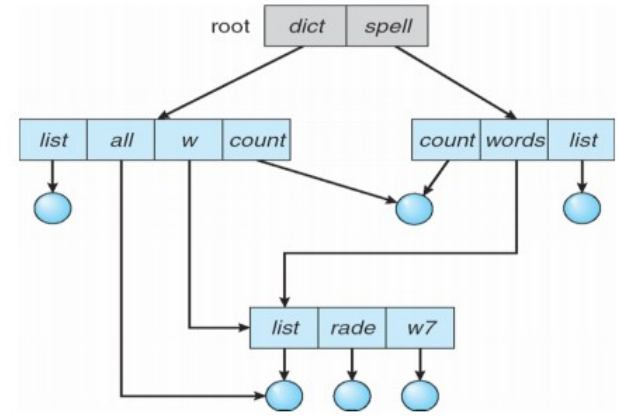
Si dict borra list -> quedan punteros solitarios

Soluciones:

- *Punteros hacia atrás, así se pueden borrar todos los punteros. Los registros de tamaño variable son un problema.*
- *Punteros hacia atrás usando una organización "cadena margarita".*
- *Contador de entradas de archivo.*

Nueva entrada en el directorio:

- *Link – otro nombre (puntero) a un archivo existente.*
- *Resuelva el link – siga el puntero hasta localizar el archivo.*



Grafo General de Directorio

Uno de los problemas más graves que afectan a la utilización de una estructura en forma de grafo acíclico consiste en garantizar que no exista ningún ciclo. Si comenzamos con un directorio en dos niveles y permitimos a los usuarios crear subdirectorios, obtendremos un directorio con estructura de árbol. Es fácil darse cuenta de que la simple adición de nuevos archivos y subdirectorios a una estructura de árbol preserva la naturaleza y la estructura de árbol de ese directorio.

Sin embargo, si añadimos enlaces a un directorio existente con estructura de árbol, la estructura de árbol se destruye, dando como resultado una estructura de grafo simple.

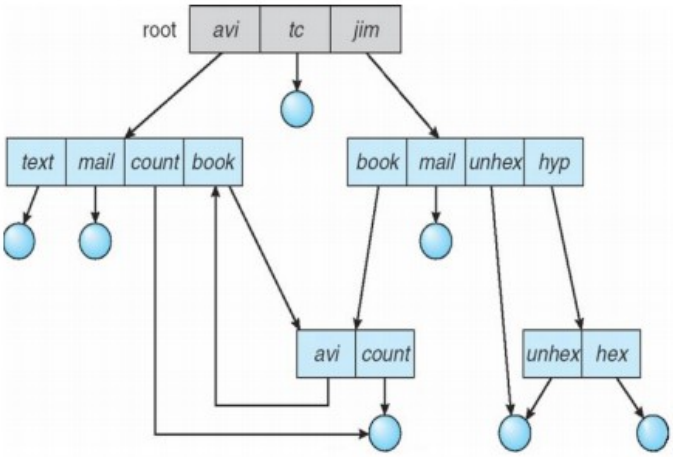
Necesitamos poder evitar recorrer las secciones compartidas de un grafo cíclico dos veces, principalmente por razones de rendimiento. Inclusive, un algoritmo mal diseñado podría provocar un bucle infinito que estuviera explorando el ciclo continuamente, sin terminar nunca.

¿Cómo se garantiza que no haya ciclos?

1. Permita enlaces (links) a archivos y no a subdirectorios
2. “Garbage collection”
3. Cada vez que se agrega un nuevo enlace (link) se usa un algoritmo de detección de ciclos para determinar si está bien.

Un problema similar existe cuando tratamos de determinar si un archivo puede ser borrado. Con las estructuras de directorio en grafo acíclico, un valor de 0 en el contador de referencia significaría que ya no hay más referencias al archivo o directorio, y que ese archivo puede ser borrado. Sin embargo, si existen ciclos, el contador de referencias puede no ser 0 incluso aunque ya no sea posible hacer referencia a un directorio de archivo. Esta anomalía resulta de la posibilidad de auto-referencia en la estructura de directorio. Una solución consiste en permitir enlaces a archivos y no a subdirectorios. Otra solución se basaría en utilizar un esquema de recolección de memoria para determinar cuándo se ha borrado la última referencia y que, en consecuencia, puede reasignarse el espacio de disco.

La principal dificultad es la de evitar que aparezcan ciclos a medida que se añaden nuevos enlaces a la estructura. *¿Cómo podemos saber si un nuevo enlace va a completar el ciclo?* Existen algoritmos para terminar la existencia de ciclos en los grafos; sin embargo, estos algoritmos son muy costosos desde el punto de vista computacional, especialmente cuando el grafo se encuentra almacenado en disco. Un algoritmo más simple en el caso especial de directorios y enlaces consiste en ignorar los enlaces durante el recorrido de directorios. De este modo, se evitan los ciclos sin necesidad de efectuar ningún procesamiento adicional.



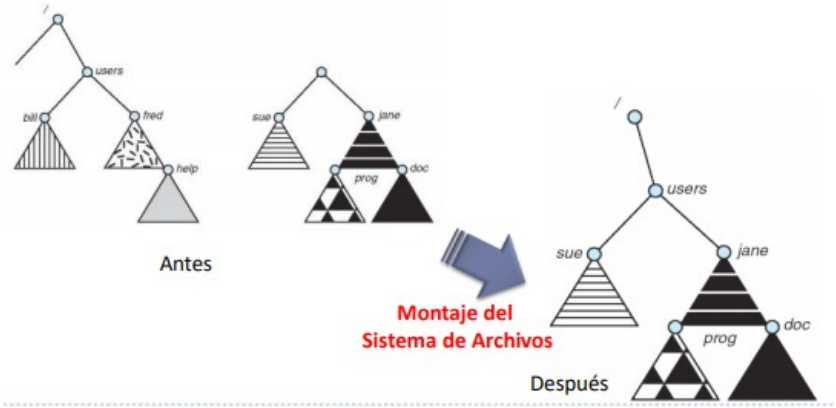
Montaje de Sistema de Archivos (Mounting)

Un sistema de archivos debe montarse para poder estar disponible para los procesos del sistema.

El proceso de montaje es bastante simple. Al sistema operativo se le proporciona el nombre del dispositivo y el punto de montaje que es la ubicación dentro de la estructura de archivos a la que hay que conectar el sistema de archivos que está montando. Normalmente, el punto de montaje será un directorio vacío.

A continuación el sistema operativo verifica que el dispositivo contiene un sistema de archivos válido. Para ello, pide al controlador de dispositivo que lea el directorio de dispositivo y verifique que ese directorio tiene el formato esperado. Finalmente, el sistema operativo registra en su estructura de directorios que hay un sistema de archivo montado en el punto de montaje especificado.

Un sistema de archivos sin montar es montado en un punto de montaje (mount point).



Archivos Compartidos

Cuando un sistema operativo tiene múltiples usuarios, las cuestiones relativas a la compartición de archivos, a la denominación de archivos y a la protección de archivos cobran una gran importancia. Dada una estructura de directorio que permite que los usuarios compartan archivos, el sistema debe adoptar un papel de mediador en lo que a la compartición de archivos respecta. El sistema puede permitir que un usuario acceda a los archivos de otros usuarios de manera predeterminada, o puede por el contrario requerir que los usuarios concedan explícitamente el acceso a sus archivos. La acción de compartir debe ser hecha por medio de un esquema de protección.

Para implementar la compartición y los mecanismos de protección, el sistema debe mantener más atributos de los archivos y de los directorios. La mayoría de los sistemas han optado por usar el concepto de propietario de un archivo o directorio y el concepto de grupo. El propietario es el usuario que puede cambiar los atributos y conceder el acceso y que dispone del máximo grado de control sobre el archivo. El atributo de grupo define un subconjunto de usuarios que pueden compartir el acceso al archivo.

Cuando un usuario solicita realizar una operación sobre un archivo, se puede comparar el ID del usuario con el atributo de propietario para determinar si el usuario solicitante es el propietario del archivo. De la misma manera, pueden compararse los identificadores de grupo. El resultado indicara que permisos son aplicables. A continuación, el sistema aplicará dichos permisos a la operación solicitada y la autorizara o negara.

En sistemas distribuidos los archivos pueden ser compartidos a través de la red. Network File System (NFS) es un método común de compartir archivos distribuidos. Los Users IDs identifican usuarios, admiten permisos y protección por usuarios. Los Groups IDs admite agrupar usuarios en grupos, permitiendo asignar al mismo derechos de acceso.

Archivos Compartidos – Semántica de Consistencia

La semántica de consistencia especifica cómo pueden acceder simultáneamente a un archivo múltiples usuarios del sistema. En particular, especifica cuando las modificaciones que un usuario realice en los datos serán observados por parte de los otros usuarios. La semántica consistencia está directamente relacionada con los algoritmos de sincronización de procesos.

Por otro lado el sistema **UNIX utiliza la siguiente semántica de coherencia:**

- Las escrituras en un archivo abierto por parte de un usuario son visibles inmediatamente para los otros usuarios que hayan abierto ese archivo.
- Un modo de compartición permite a los usuarios compartir el puntero de ubicación actual dentro del archivo. Así, el incremento de ese puntero por parte del usuario afectara a todos los usuarios que estén compartiendo el archivo.

En la semántica UNIX, cada archivo está asociado con una única imagen física a la que se accede en forma de recurso exclusivo.

En el **sistema de archivos Andrew se implementó una semántica** muy compleja para compartir archivos que consta de lo siguiente:

- Las escrituras en un archivo abierto por parte de un usuario no son visibles inmediatamente para los restantes usuarios que hayan abierto el mismo archivo.
- Una vez que se cierra un archivo, los cambios realizados en él son visibles únicamente en las sesiones que den comienzo posteriormente. Las instancias ya abiertas del archivo no reflejarán dichos cambios.

De acuerdo con esta semántica, un archivo puede estar temporalmente asociado con varias imágenes al mismo tiempo. En consecuencia, se permite que múltiples usuarios realicen accesos concurrentes tanto de lectura como de escritura en sus propias imágenes del archivo, sin ningún retardo.

Semántica de sesión: AFS tiene una semántica de sesión

- Las escrituras solo son visibles después de que la sesión termina.

Semántica de archivos compartidos inmutables

Una vez que un archivo es declarado como compartido por su creador, no puede ser modificado. Un archivo inmutable tiene dos propiedades clave: su nombre no puede reutilizarse y su contenido no puede ser modificado. Por lo tanto, el hecho de que un archivo sea inmutable significa que su contenido esta fijo. La implementación de esta semántica en un sistema distribuido es bastante simple, ya que el mecanismo de compartición es disciplinado.

Protección

Los sistemas que no permiten el acceso a los sistemas de otros usuarios no necesitan ninguna protección. Así, podríamos proporcionar una protección completa simplemente prohibiendo el acceso. Alternativamente, podríamos proporcionar un acceso libre sin ninguna protección. Ambas técnicas son demasiado extremas para poder utilizarlas de forma general; lo que se necesita es un acceso controlado -> El creador/dueño del archivo debería poder controlar que acciones pueden realizarse con el archivo y quienes las pueden realizar. Se habla entonces de acceso controlado.

Los mecanismos de protección proporcionan este acceso controlado limitando los tipos de accesos a archivo que puedan realizarse. Podemos controlar varios tipos de operaciones diferentes, entre ellas:

- **Lectura.** Lectura de un archivo.
- **Escritura.** Escritura o reescritura de un archivo.
- **Ejecución.** Carga del archivo en memoria y ejecución del mismo.

- **Adición.** Escritura de la nueva información al final del archivo.
- **Borrado.** Borrado del archivo y liberación del espacio para su posible reutilización.
- **Listado.** Listado del nombre y atributos del archivo.

Control de acceso

La técnica más común para resolver el problema de la protección consiste en hacer que el acceso dependa de la identidad del usuario. Los diferentes usuarios pueden necesitar diferentes tipos de acceso (read, write o execute) a un archivo o directorio. El esquema más general para implementar un acceso dependiente de la identidad consiste en asociar con cada archivo y directorio una **lista de control de acceso (ACL, access-control list)** que especifique los nombres del usuario y los tipos de acceso que se permiten para cada uno. Cuando un usuario solicita acceder a un archivo concreto, el sistema operativo comprueba la lista de acceso asociada con dicho archivo; si dicho usuario está incluido en la lista para el tipo de acceso solicitado, se permite el acceso. En caso contrario, se producirá una violación de la protección y se denegará al trabajo de usuario el acceso al archivo.

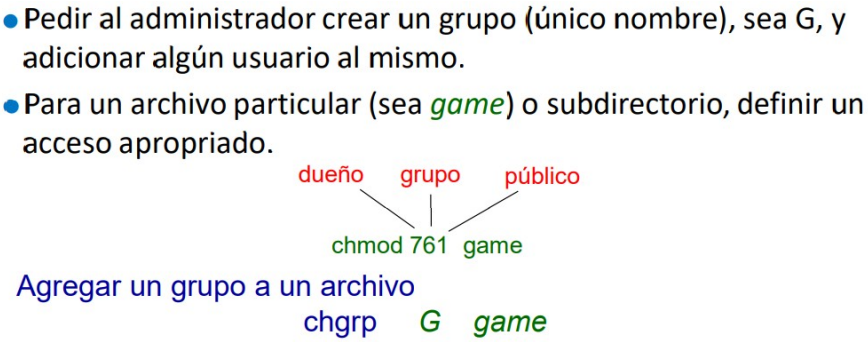
Esta técnica tiene la ventaja de permitir la implementación de complejas metodologías de acceso. El problema principal con las listas de acceso es su longitud. Si queremos permitir que todo el mundo lea un archivo, deberemos enumerar a todos los usuarios que disponen de ese acceso de lectura. La cual es una tarea tediosa y requiere que la entrada de directorio tenga tamaño variable ya que no sabemos de antemano la lista de usuarios del sistema.

Estos problemas pueden resolverse utilizando una versión condensada de la lista de acceso. Para condensar la longitud de la lista de control de acceso, muchos sistemas clasifican a los usuarios en tres grupos, en lo que respecta con cada archivo:

- ⇒ Propietario: el usuario que creo el archivo será su propietario.
- ⇒ Grupo: Un conjunto de usuarios que están compartiendo el archivo y necesitan un acceso similar al mismo es un grupo, o grupo de trabajo.
- ⇒ Universo. Todos los demás usuarios del sistema constituyen el universo.

La técnica reciente más común consiste en combinar las listas de control de acceso con el esquema más general de control de acceso para lograr una granularidad más fina.

Para que este esquema funcione adecuadamente, deben controlarse estrechamente tanto los permisos como las listas de control de acceso. Este control puede llevarse acabo de varias formas distintas. Por ejemplo, en el sistema UNIX, los grupos sólo pueden ser creados y modificados por el administrador de la instalación o cualquier superusuario.



Sistemas de Archivos Implementación - Módulo 11

Estructura del Sistema de Archivos

Los discos constituyen el principal tipo de almacenamiento secundario para mantener sistemas de archivos. Tienen dos características que los convierten en un medio conveniente para el almacenamiento de múltiples archivos:

1. El disco puede ser reescrito de manera directa; es posible leer un bloque del disco, modificar un bloque y volverlo a escribir en el mismo lugar.
2. Con un disco se puede acceder directamente a cualquier bloque de información que contenga. Por tanto, resulta simple acceder a cualquier archivo de forma secuencial aleatoria y el conmutar de un archivo a otro solo requiere mover los cabezales de lectura-escritura y esperar a que el disco termine de rotar.

En lugar de transferir un byte cada vez, las transferencias de E/S entre la memoria y el disco se realizan en unidades de bloques, para mejorar la eficiencia de E/S. Cada bloque tiene uno o más sectores.

Para proporcionar un acceso eficiente y cómodo al disco, el sistema operativo impone uno o más **sistemas de archivos,** con los que los datos pueden almacenarse, localizarse y extenderse fácilmente. Un sistema de archivos acarrea dos problemas de diseño bastante diferentes. El primer problema es definir qué aspecto debe tener el sistema de archivos para el usuario. Esta tarea implica definir un archivo y sus atributos, las operaciones definidas sobre los archivos y la estructura de directorio utilizada para organizar los archivos. El segundo problema es crear algoritmos y estructuras de datos que permitan mapear el sistema lógico de archivos sobre los dispositivos físicos de almacenamiento secundario.

El sistema de archivos está organizado en capas. Cada nivel de diseño utiliza las funciones de los niveles inferiores para crear nuevas funciones que serán, a su vez, utilizadas por los niveles superiores a este.

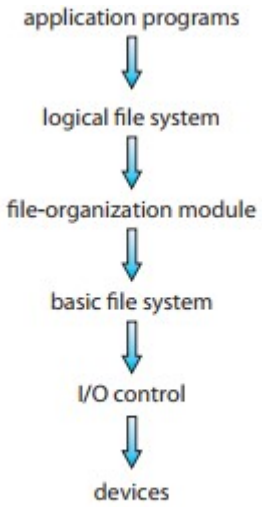


Figure 12.1 Layered file system.

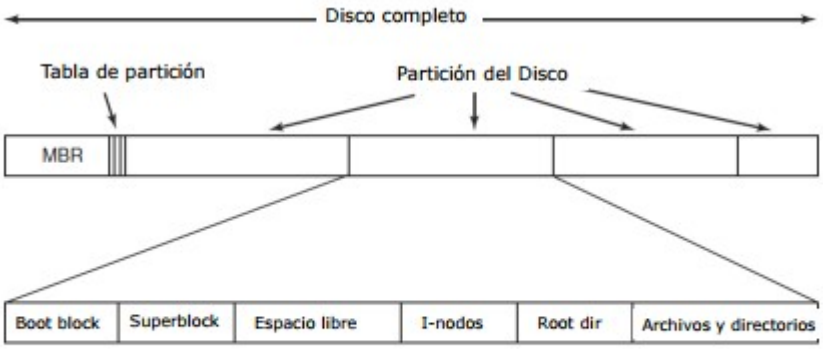
El nivel más bajo, el control de E/S, está compuesto por controladores de dispositivo y rutinas de tratamiento de interrupción, que se encargan de transferir la información entre la memoria principal y el sistema de disco.

El **sistema básico de archivos** sólo necesita enviar comandos genéricos al controlador de dispositivo apropiado, con el fin de leer y escribir bloques físicos en el disco. Cada bloque físico se identifica mediante su dirección de disco numérica.

El **módulo de organización de archivos** tiene conocimiento acerca de los archivos y de sus bloques lógicos, así como de sus bloques físicos. Considerando el tipo de asignación de archivos utilizado y la ubicación del archivo, el módulo de organización de archivos puede traducir las direcciones lógicas de bloque a direcciones físicas de bloque, que serán las que envíe al sistema básico de archivos para que realice las necesarias transferencias. El módulo de organización de archivos incluye también el gestor de espacio libre, que controla los bloques no asignados.

Finalmente, el sistema lógico de archivos gestiona la información de metadatos. Los metadatos incluyen toda la estructura del sistema de archivos, excepto los propios datos (es decir, el propio

contenido del archivo). El sistema lógico de archivos gestiona la estructura de directorio para proporcionar al módulo de organización de archivos la información que este necesita, a partir de un nombre de archivo simbólico. Este nivel mantiene la estructura de los archivos mediante bloques de control de archivo. Un bloque de control de archivo (FCB, file-control block) contiene información acerca del archivo, incluyendo su propietario, los permisos y la ubicación del contenido del archivo. El sistema lógico también es el responsable de las tareas de protección y seguridad.



Ejemplo de un posible diseño

Estructuras de Archivo

- ✓ Unidad lógica de almacenamiento
- ✓ Colección de información relacionada.

Se utilizan varias estructuras en disco y en memoria para implementar un sistema de archivos. Estas estructuras varían dependiendo del sistema operativo y del sistema de archivos, aunque algunos principios son de aplicación general:

- ⇒ **Un bloque de control de arranque** puede contener la información que se necesita para iniciar un sistema operativo a partir de dicho volumen. Si dicho disco no contiene un sistema operativo, este bloque puede estar vacío.
- ⇒ Un **bloque de control de volumen** contiene detalles acerca del volumen, tales como el número de bloques que hay en la partición, el tamaño de los bloques, el número de bloques libres y los punteros de bloques libres, así como un contador de FCB libres y punteros a FCB.
- ⇒ Se utiliza una estructura de directorios por cada sistema de archivos para organizar los archivos.
- ⇒ Un bloque FTB por cada archivo contiene numerosos detalles acerca del archivo, incluyendo los permisos correspondientes, el propietario, el tamaño y la ubicación de los bloques de datos.

👁 File Control Block (FCB): estructura de almacenaje consistente de información sobre el archivo.

Estructura del Sistema de Archivos en Memoria

La información almacenada en memoria se utiliza tanto para la gestión de un sistema de archivos como para la mejora del rendimiento mediante mecanismos de caché.
Una tabla de montaje en memoria contiene información acerca de cada volumen montado. Una caché de la estructura de directorios en memoria almacena la información relativa a los directorios que se han accedido recientemente.

La tabla global de archivos abiertos contiene una copia del FCB de cada archivo abierto. La tabla de archivos abiertos de cada proceso contiene un puntero a la entrada apropiada de la entrada global de archivos abiertos.
Para crear un nuevo archivo, un programa de aplicación llama al sistema lógico de archivos. El sistema lógico de archivos asigna un nuevo FCB. El sistema lee entonces el directorio apropiado en la memoria, lo actualiza con el nombre del nuevo archivo y el nuevo FCB y lo vuele a escribir en el disco. Una vez creado el archivo, ya podemos utilizarlo para E/S. Pero primero, sin embargo, debemos abrirlo. La llamada open() pasa un nombre de archivo al sistema de archivos. Esta llamada primero busca en la tabla global de archivos abiertos para ver si el archivo está siendo ya utilizado por otro proceso. En caso afirmativo, se crea una entrada en la tabla de archivos abiertos del proceso que apunte a la tabla global de archivos abiertos existente. Cuando el archivo está abierto, se busca en la estructura de directorios para encontrar el nombre del archivo indicado. Una vez encontrado el archivo, el FCB se copia en la tabla global de archivos abiertos existente en la memoria. Esta tabla no sólo almacena el FCB, sino que también controla el número de procesos que han abierto el archivo. A continuación, se crea una entrada en la tabla de archivos abiertos del proceso, con un puntero a la entrada de la tabla global de archivos abiertos y algunos otros campos de información. La llamada a open() devuelve un puntero a la entrada apropiada de la tabla de archivos abiertos del proceso; todas las operaciones de archivo se realizan a partir de ahí mediante este puntero. El nombre del archivo que se proporciona a la caché para futuras operaciones de apertura se conoce como descriptor de archivo.
Cuando un proceso cierra el archivo, se elimina la entrada de la tabla del proceso y se decrementa el contador de aperturas existente en la tabla global. Cuando todos los usuarios que hayan abierto el archivo lo cierren, los metadatos actualizados se copian en la estructura de directorio residente en el disco y se elimina la entrada de la tabla global de archivos abiertos.

Implementación de Directorio

- Lista lineal: Es el método más simple para implementar un directorio. Consiste en utilizar una lista lineal de nombres de archivos, con punteros a los bloques de datos. Este método es simple de programar, pero requiere mucho tiempo de ejecución ya que para localizar un archivo, se requiere realizar una búsqueda lineal.
- Tabla hash: Con este método, se almacenan las entradas de directorio en una lista lineal, pero también se utiliza una estructura de datos hash. La tabla toma un valor calculado a partir del nombre del archivo y devuelve un puntero a la ubicación de dicho nombre de archivo dentro de la lista lineal. Por tanto, puede reducir enormemente el tiempo de búsqueda en el directorio. La inserción y el borrado son también bastante sencillas, aunque es necesario tener en cuenta la posible aparición de colisiones, que son aquellas situaciones en las que dos nombres de archivo proporcionan, al aplicar la función hash, la misma ubicación dentro de la lista. Las principales dificultades asociadas con las tablas hash son que su tamaño es, por regla general, fijo y que la función hash depende de dicho tamaño.

Métodos de Alocación

Casi siempre existirán múltiples archivos almacenados en un mismo disco. El principal problema es cómo asignar el espacio a esos archivos de forma rápida. Hay tres métodos principales de asignación del espacio de disco que se utilizan de manera común: **alocación contigua, enlazada e indexada.**

1. Alocación Contigua

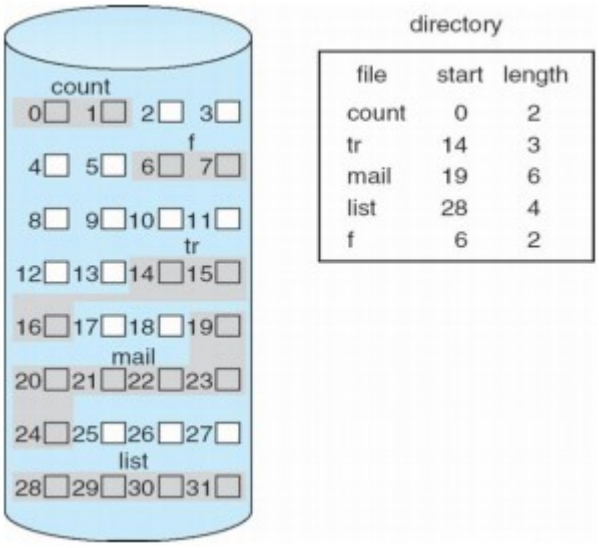
La alocación contigua requiere que cada archivo ocupe un conjunto de bloques contiguos en el disco. Las direcciones de disco definen una ordenación lineal del disco.

La asignación contigua de un archivo está definida por la asignación de disco del primer bloque y por la longitud del archivo, en unidades de bloques. La entrada de directorio de cada archivo indicara la dirección del bloque de inicio y la longitud de área asignada al archivo.

Acceder a un archivo que haya sido asignado de manera contigua resulta sencillo. Para el acceso secuencial, el sistema de archivos recuerda la dirección de disco del último bloque al que se haya hecho referencia y leerá el siguiente bloque cuando sea necesario. Para el acceso directo al bloque i de un archivo que comience en el bloque b, podemos acceder inmediatamente al bloque b+i. Así, la asignación contigua permite el acceso aleatorio.

Sin embargo, la asignación contigua también tiene sus problemas. Una de las dificultades estriba en encontrar espacio para un nuevo archivo. El problema de la asignación contigua puede verse como un caso concreto del problema general de asignación dinámica del almacenamiento y que se refiere a como satisfacer una solicitud de tamaño n a partir de una lista de huecos libres. Las estrategias más comunes para seleccionar un hueco libre a partir de un conjunto de huecos disponibles son la de primer ajuste y mejor ajuste aunque sufren de fragmentación externa. Dependiendo de la cantidad total del espacio de almacenamiento en disco y del tamaño medio de los archivos, la fragmentación externa puede ser un problema grave o demasiado importante.

Otro problema de los mecanismos de asignación contigua es determinar cuánto espacio se necesita para un archivo. En el momento de crear el archivo, será necesario determinar la cantidad total de espacio que vamos a necesitar y asignar esa cantidad total. Si asignamos demasiado poco espacio a un archivo, podemos encontrarnos con que no puede ampliarse. Especialmente con las estrategias de asignación de mejor ajuste, puede que el espacio situado a ambos lados del archivo ya esté siendo utilizado, por tanto, no podemos ampliar el tamaño del archivo sin desplazarlo a otro lugar. Entonces, existen dos posibilidades. La primera es terminar el programa de usuario emitiendo un error apropiado o localizar un hueco de mayor tamaño, copiar el contenido del archivo al nuevo espacio y liberar el anterior.

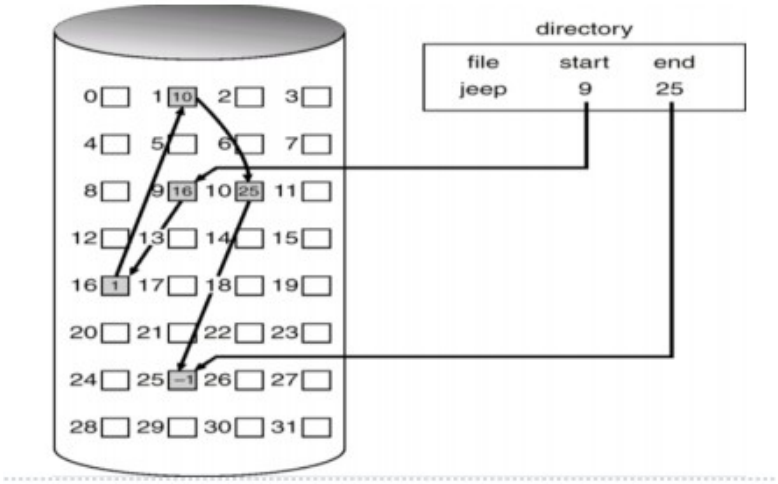


2. Alocación Enlazada

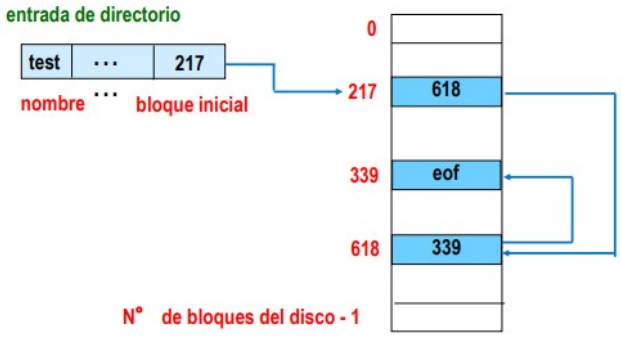
El mecanismo de asignación enlazada resuelve todos los problemas de asignación contigua.

Con la asignación enlazada, cada archivo es una lista enlazada de bloques de disco, pudiendo estar dichos bloques dispersos por todo el disco. El directorio contiene un puntero al primer y al

último bloque de cada archivo. Para leer un archivo, simplemente leemos los bloques siguiendo los punteros de un bloque a otro.
No hay ninguna fragmentación externa con asignación enlazada y puede utilizarse cualquier de los bloques de la lista de bloques libres para satisfacer una solicitud. No es necesario declarar el tamaño del archivo en el momento de crearlo y el archivo puede continuar creciendo mientras existan bloques libres. En consecuencia, nunca es necesario compactar el espacio en disco.
Sin embargo, este mecanismo también tiene sus desventajas. El principal problema es que sólo se lo puede utilizar de manera efectiva para los archivos de acceso secuencial. Para encontrar el bloque i-esimo de un archivo, podemos comenzar al principio de dicho archivo y seguir los punteros hasta llegar al bloque i-esimo. Cada acceso a un puntero requiere una lectura de disco y algunos de ellos requerirán un reposicionamiento de cabezales. En consecuencia, resulta muy poco eficiente tratar de soportar una funcionalidad de acceso directo para los archivos de alocación enlazada.
Otra desventaja es el espacio requerido para los punteros. Otro problema de la asignación enlazada es la fiabilidad ya que los punteros se pueden dañar arruinando la estructura.



Una variante importante del mecanismo de asignación enlazada es la que se basa en el uso de una tabla de asignación de archivos (FAT). Este método simple, pero eficiente, de asignación del espacio del disco se utiliza en los sistemas operativos MS-DOS y OS/2.



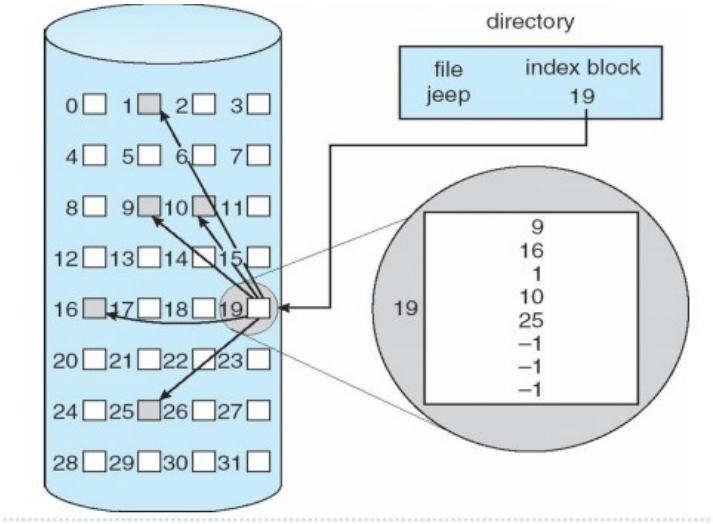
3. Asignación indexada

Bajo el método de alocación enlazada, sino se utiliza una FAT, no se puede soportar el acceso directo de manera eficiente → El mecanismo de alocación indexada resuelve este problema agrupando todos los punteros en una única ubicación: el bloque de índice.
Cada archivo tiene su propio bloque de índice, que es una matriz de direcciones de bloques de disco. La entrada i-esima del bloque de índice apunta al bloque i-esimo del archivo. El directorio

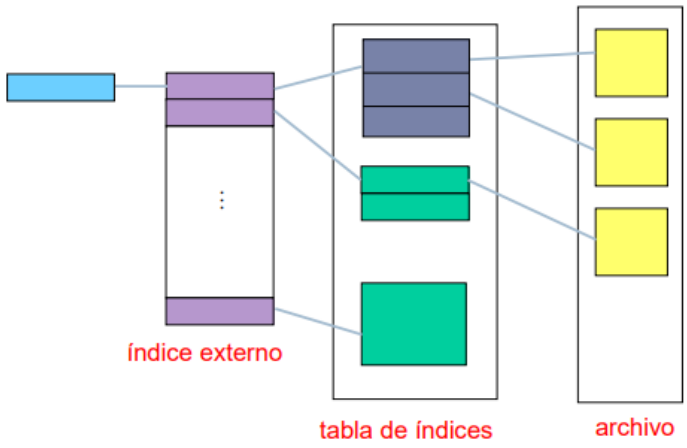
contiene la información del bloque de índice. Para localizar y leer el bloque i-esimo, utilizamos el puntero contenido en la entrada i-esima del bloque de índice.
El mecanismo de asignación indexada soporta el acceso directo, sin sufrir el problema de la fragmentación externa, ya que puede utilizarse cualquier bloque de disco para satisfacer una solicitud de espacio.
Sin embargo, el mecanismo de asignación indexada sí que sufre el problema de desperdicio de espacio. El espacio adicional requerido para almacenar los punteros del bloque de índice es, generalmente, mayor que el que se requiere en el caso de la asignación enlazada. Este punto nos plantea la cuestión de cuál debe ser el tamaño del bloque de índice. Cada archivo debe tener un bloque de índice, así que ese bloque deberá ser lo más pequeño posible. Sin embargo, si el bloque de índice es demasiado pequeño, no podrá almacenar suficientes punteros para un archivo de gran tamaño y será necesario implementar un mecanismo para resolver este problema. Entre los mecanismos que pueden utilizarse para ello se encuentran los siguientes:

- Esquema enlazado: Cada bloque de índice ocupa normalmente un bloque de disco. Por tanto, puede leerse y escribirse directamente. Para que puedan existir archivos de gran tamaño, podemos enlazar varios bloques de índice.
- Índice multinivel: Una variante de la representación enlazada consiste en utilizar un bloque de índice de primer nivel para apuntar a un conjunto de bloques de índice de segundo nivel, que a su vez apuntaran a los bloques del archivo. Para acceder a un bloque, el sistema operativo utiliza el índice de primer nivel para encontrar un bloque de índice de segundo nivel y luego emplea dicho bloque para hallar el bloque de datos deseado. Esta técnica podría ampliarse hasta un tercer o cuarto nivel, dependiendo del tamaño de archivo que se desee.
- Esquema combinado: Otra alternativa, consiste en mantener, por ejemplo, los primeros 15 punteros del bloque de índice en el nodo del archivo. Los primeros 12 de estos punteros hacen referencia a bloques directos, es decir, contienen la dirección de una serie de bloques que almacenan datos del archivo. De este modo, los datos para los archivos de pequeño tamaño no necesitan un bloque de índice separado. Los siguientes tres punteros hacen referencia a bloques indirectos. El primero apunta a un bloque indirecto de un nivel, que es un bloque de índice que no contiene datos sino las direcciones de otros bloques que almacenan los datos. El segundo puntero hace referencia a un bloque doblemente indirecto, que contiene la dirección de un bloque que almacena las direcciones de otras series de bloques que contienen punteros a los propios bloques de datos. El último puntero contiene la dirección de un bloque triplemente indirecto.

Los esquemas de asignación indexados sufren de los mismos problemas de rendimiento que el mecanismo de asignación enlazada. Específicamente, los bloques de índice pueden almacenarse en memoria caché, pero los bloques de datos pueden estar distribuidos por todo un volumen.



Alocación indexada



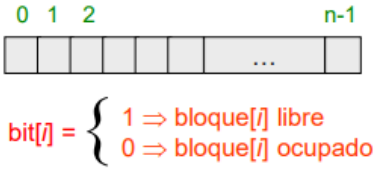
Alocación indexada-Mapeo

Administración de Espacio Libre

Puesto que el espacio de disco está limitado, necesitamos reutilizar el espacio de los archivos borrados para los nuevos archivos, siempre que sea posible. Para controlar el espacio libre del disco, el sistema mantiene una **lista de espacio libre**. La lista de espacio libre indica todos los bloques de disco libres, aquellos que no están asignados a ningún archivo o directorio. Para crear un archivo, exploramos la lista de espacio libre hasta obtener la cantidad de espacio requerida y asignamos ese espacio al nuevo archivo. A continuación, este espacio se elimina de la lista de espacio libre. Cuando se borra un archivo, su espacio de disco se añade a la lista de espacio libre.

Vector de bits

Recientemente, la lista de espacio libre se implementa como un mapa de bits o vector de bits. Cada bloque está representado por 1 bit. Si el bloque esta libre, el bit será igual a 1; si el bloque está asignado, el bit será 0.



La principal ventaja de este enfoque es su relativa simplicidad y la eficiencia que permite a la hora de localizar el primer bloque libre o n bloques libres consecutivos en el disco. Una técnica para localizar el primer bloque libre en un sistema que utilice un lector de bits para asignar el espacio de disco consiste en comprobar secuencialmente cada palabra del mapa de bits para ver si dicho valor es distinto de 0, ya que una palabra con valor 0 tendrá todos los bits iguales a 0 y representará un conjunto de bloques asignado. Cuando se encuentra una palabra distinta de 0 se explora en búsqueda del primero bit 1, que corresponderá con la ubicación del primer bloque libre. El cálculo del número de bloque será:

(número de bits por palabra) x (número de palabras de valor 0) + offset del primer bit 1

Desafortunadamente, los vectores de bits son ineficientes a menos que se mantenga el vector completo en la memoria principal. Mantener ese vector en la memoria principal es posible para discos de pequeño tamaño, pero no necesariamente para los discos de tamaño grande.

Lista enlazada

Otra técnica para la gestión del espacio libre consiste en enlazar todos los bloques de disco libres, manteniendo un puntero al primer bloque libre en ubicación especial del disco y almacenándolo en memoria cache. A pesar de que no malgasta espacio, este esquema es poco eficiente; para recorrer la lista, debemos leer cada bloque, lo que requiere un tiempo sustancial de E/S.

Agrupamiento

Una modificación de la técnica basada en la lista de espacio libre consiste en almacenar las direcciones de n bloques libres en el primer bloque libre. Los primeros n-1 de estos bloques estarán realmente libres. De este modo, podrán encontrarse rápidamente las direcciones de un gran número de bloques libres, a diferencia del caso en que se utilice la técnica estándar de lista enlazada.

Recuento

Otra técnica consiste en aprovechar el hecho de que, generalmente, puede asignarse o liberarse simultáneamente varios bloques contiguos, particularmente cuando se asigna el espacio mediante el algoritmo de asignación contigua o mediante un mecanismo de agrupación cluster. Así, en lugar de mantener la lista de n direcciones de bloques de disco libres, podemos mantener la dirección del primer bloque libre y el número n de bloques libres contiguos que siguen a ese primer bloque. Cada entrada en la lista de espacio libre estará entonces compuesta por una dirección de disco y un contador. Esto provoca que la lista global sea más corta, siempre y cuando el valor del contador sea mayor que 1.

Eficiencia y Desempeño

El uso eficiente del espacio de disco depende en gran medida de los algoritmos de asignación de disco y de directorio que se utilicen. También debemos considerar los tipos de datos que normalmente se almacenan en la entrada de directorio de un archivo. El resultado de mantener esta información es que, cada vez que se realiza una operación con el archivo, puede ser necesario escribir en uno de los campos de la estructura de directorio.

Incluso después de haber seleccionado los algoritmos básicos del sistema de archivos, podemos mejorar las prestaciones de diversas maneras:

- Algunos sistemas mantienen una sección separada de memoria principal, reservándola para caché de búfer en la que se almacenan los bloques bajo la suposición de que en breve volverán a ser utilizados de nuevo.
- Otros sistemas almacenan en caché los datos de archivos utilizando una caché de páginas. La cache de páginas utiliza técnicas de memoria virtual para almacenar en cache los datos de los archivos en forma de páginas, en lugar de almacenar en bloques de sistema de archivos, lo cual resulta mucho más eficiente.
- Algunos sistemas optimizan su cache de páginas utilizando diferentes algoritmos de sustitución, dependiendo del tipo de acceso del archivo. Para los archivos que se lean o escriban secuencialmente, sus páginas no se sustituirán en orden LRU, porque las páginas más recientemente utilizadas serán las que se utilicen más tarde, o quizá no lleguen a utilizarse nunca. En lugar de ello, el acceso secuencial puede optimizarse mediante ciertas técnicas conocidas como **free-behind y read-ahead**.
    - **Free-behind** elimina una página del búfer en cuanto se solicita la página siguiente; las páginas anteriores no volverán, probablemente, a utilizarse de nuevo, así que representan un desperdicio en el búfer.
    - Con el mecanismo de **read-ahead**, cuando se solicita una página, se leen y almacenan en caché tanto esa página como varias páginas sucesivas. Dichas páginas es bastante probable que se soliciten después de procesar la página actual.

Recuperación

Los archivos y directorios se mantienen tanto en memoria principal como en disco, y debe tenerse cuidado para que los fallos del sistema no provoquen una pérdida de datos o una incoherencia en los mismos.

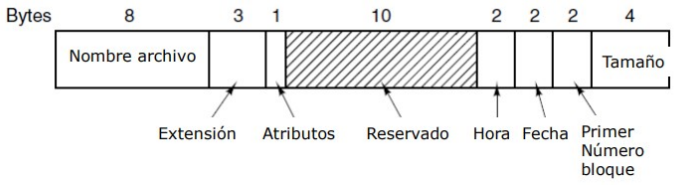
El **comprobador de coherencia** compara los datos de la estructura de directorios con los bloques de datos del disco y trata de corregir todas las incoherencias que detecte. Los algoritmos de asignación y gestión del espacio libre dictan los tipos de problemas que el comprobador puede tratar de detectar y dictan también el grado de éxito que el comprobador puede tener al realizar esa tarea.

Los discos magnéticos fallan en ocasiones y es necesario tener cuidado para garantizar que los datos perdidos debido a esos fallos no se pierdan para siempre. Con este fin, pueden utilizarse programas del sistema para realizar una copia de seguridad de los datos (backup) del disco en otro dispositivo de almacenamiento (cinta magnética, optical, etc). La recuperación de la pérdida de un archivo individual o de un disco completo puede ser entonces, simplemente, una cuestión de restaurar los datos a partir de la copia de seguridad.

Sistema de archivos de Red de Sun (NFS)

Es una implementación y una especificación es un sistema de software para acceder a archivos remotos a través de LANs (o WANs). La implementación es parte de los sistemas operativos Solaris y SunOS que corre sobre estaciones de trabajo Sun usando un protocolo no confiable datagrama (protocolo UDP/IP) y Ethernet.

Sistema de archivos – MS-DOS



Entrada de directorio

Tam. Bloque	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Máximo tamaño de la partición para diferentes tamaños de bloques

Sistema de Archivos Almacenaje Secundario - Módulo 12

Revisión

Los discos magnéticos proporcionan la parte principal del almacenamiento secundario en los modernos sistemas informáticos. Desde el punto de vista conceptual, los discos son relativamente simples. Cada plato tiene una forma circular plana, como un CD. Las dos superficies de cada plato están recubiertas de un material magnético. La información se almacena grabándola magnéticamente sobre los platos.

Un cabezal de lectura-escritura vuela justo por encima de cada una de las superficies de cada plato. Los cabezales están conectados a un brazo del disco que mueve todos los cabezales como una sola unidad. La superficie de cada plato está dividida desde el punto de vista lógico en pistas

circulares, que a su vez se subdividen en sectores. El conjunto de pistas que están situadas en una determinada posición del brazo forman un cilindro. Puede haber miles de cilindros concéntricos en una unidad de disco y cada pista puede contener cientos de sectores.
Cuando se está usando el disco, un motor hace que gire a gran velocidad. La mayoría de los motores rotan entre 60 y 200 veces por segundo. La velocidad de un disco puede considerarse compuesta por dos partes diferenciadas: la velocidad de transferencia es la velocidad con la que los datos fluyen entre la unidad de disco y la computadora. El tiempo de posicionamiento, a veces denominado tiempo de acceso aleatorio, está compuesto del tiempo necesario para mover el brazo del disco hasta el cilindro deseado, denominado tiempo de búsqueda, y el tiempo requerido para el sector deseado rote hasta pasar por debajo del cabezal del disco, denominado latencia rotacional.
Puesto que el cabezal del disco vuela sobre un colchón de aire extremadamente fino, existe el peligro de que el cabezal entre en contacto con la superficie del disco. Aunque los platos del disco están recubiertos de una capa fina protectora, a veces el cabezal puede llegar a dañar la superficie magnética; este accidente se denomina aterrizaje de cabezales. Normalmente, los aterrizajes de cabezales no pueden repararse, siendo necesario sustituir el disco completo.
Un disco puede ser extraíble, lo que permite montar diferentes discos según sea necesario.
Para conectar una unidad de disco a una computadora, se utiliza un conjunto de cables denominado bus de E/S. Hay disponibles varios tipos de buses, incluyendo EIDE, ATA, SATA, USB, Fiber Channel (FC) y SCSI. Las transferencias de datos en un bus son realizadas por procesadores electrónicos especiales denominados controladoras. La controladora host es la controladora situada en el extremo del bus correspondiente a la computadora; además, en cada unidad de disco se integra una controladora de disco.

Consideraciones

- Los discos rotan n veces por segundo.
- El ritmo de transferencia.
- Tiempo de posicionamiento (tiempo de acceso al azar) -> tiempo de búsqueda + latencia rotacional.
- Un aterrizaje de cabeza tiene lugar cuando las cabezas del disco hacen contacto con la superficie del disco.

Estructura de Disco

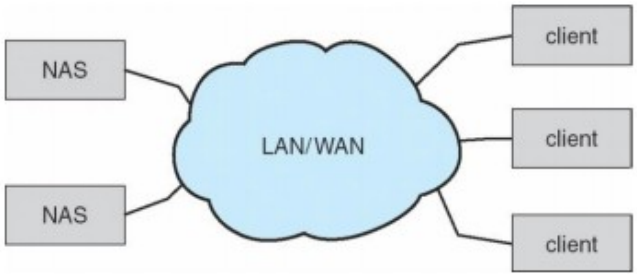
Los dispositivos de disco son vistos como un arreglo unidimensional de bloques lógicos, donde el bloque lógico es la más pequeña unidad de transferencia. Ese arreglo de bloques lógicos es mapeado secuencialmente en sectores del disco. El sector 0 es el primer sector de la primera pista sobre el cilindro más externo. El mapeo procede en orden a través de esa pista, luego el resto de las pistas en el cilindro, y luego el resto de los cilindros desde el más externo hasta el más interno.

Almacenaje Adjunto de Red

Un dispositivo de almacenamiento conectado a la red (NAS, network-attached storage) es un sistema de almacenamiento de propósito general al que se accede de forma remota a través de una red de datos. Los clientes acceden al almacenamiento conectado a la red a través de una interfaz de llamadas a procedimientos remotos, como por ejemplo NFS para los sistemas UNIX o CIFS para las máquinas Windows. Las llamadas a procedimientos remotos RPC, se realizan mediante los protocolos TCP o UDP sobre una red IP.
El almacenamiento conectado a la red proporciona una forma conveniente para que todas las computadoras de una LAN compartan un conjunto de dispositivos de almacenamiento con la misma facilidad de denominación y de acceso que si se tratara de dispositivos de

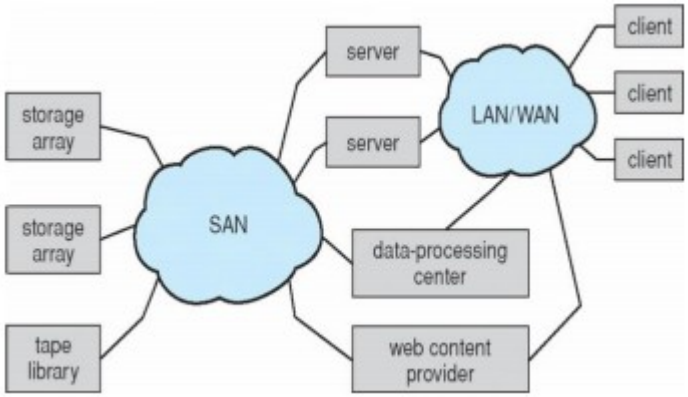
almacenamiento locales conectados al host. Sin embargo, este sistema tiende a ser menos eficiente y proporciona menos velocidad que algunas opciones de almacenamiento de conexión directa.

- ✓ NFS y CIFS son protocolos comunes.
- ✓ Un nuevo protocolo iSCSI usa IP para transportar el protocolo SCSI.



Almacenamiento en Área de Red (SAN)

Una red de almacenamiento (SAN, storage-area network) es una red privada (que utiliza protocolos de almacenamiento en lugar de protocolos de red) que conecta los servidores con las unidades de almacenamiento. La principal ventaja de una SAN radica en su flexibilidad, ya que pueden conectarse múltiples host y múltiples matrices de almacenamiento a la misma SAN y los recursos de almacenamiento pueden asignarse de forma dinámica a los hosts. Esta red es común en ambientes grandes de almacenamiento.

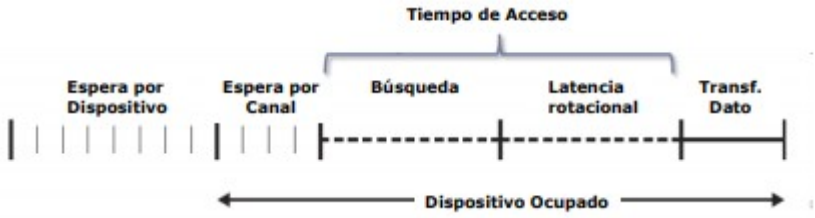


Planificación de Disco

Una de las responsabilidades del sistema operativo es la de utilizar de manera eficiente el hardware disponible. Para las unidades de disco, cumplir con esta responsabilidad implica tener un tiempo de acceso rápido y un gran ancho de banda de disco. El tiempo de acceso tiene dos componentes principales:

- ⇒ El **tiempo de búsqueda** es el tiempo requerido para que el brazo del disco mueva los cabezales hasta el cilindro que contiene el sector deseado.
- ⇒ La **latencia rotacional** es el tiempo adicional necesario para que el disco rote y sitúe el sector deseado bajo el cabezal del disco.

El ancho de banda del disco es el número total de bytes transferidos, dividido entre el tiempo total transcurrido entre la primera solicitud de servicio y la terminación de la última transferencia. Podemos mejorar tanto el tiempo de acceso como el ancho de banda planificando en el orden adecuado el servicio de las distintas solicitudes de E/S de disco. Existen varios algoritmos para planificar el servicio de los requerimientos de E/S.

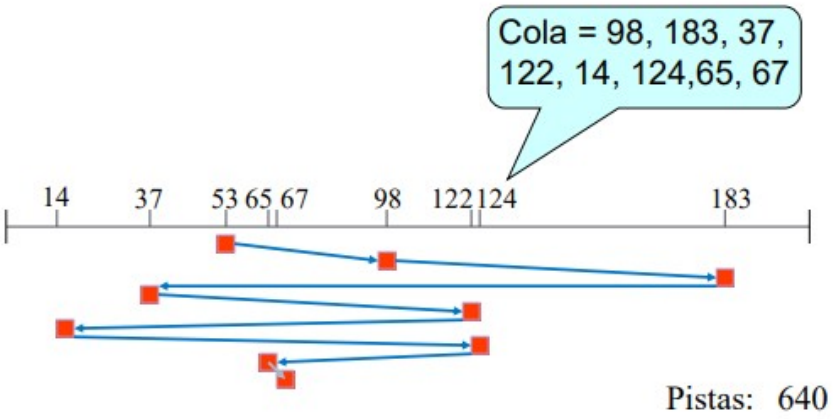


Tiempo de transferencia de E/S a Disco

Existen varios algoritmos para planificar el servicio de los requerimientos de E/S. Se ilustran los mismos con una cola de requerimientos (0-199) ➔ ***98, 183, 37, 122, 14, 124, 65, 67***
La cabeza ha resuelto el requerimiento al sector 53.

Primero en Entrar - Primero en Salir FCFS

La forma más simple de planificación de disco es, por supuesto, el algoritmo FCFS (first-cone, first-served). Este algoritmo es intrínsecamente equitativo, pero generalmente no proporciona el servicio más rápido, dado que realiza demasiados movimientos de cabezales.

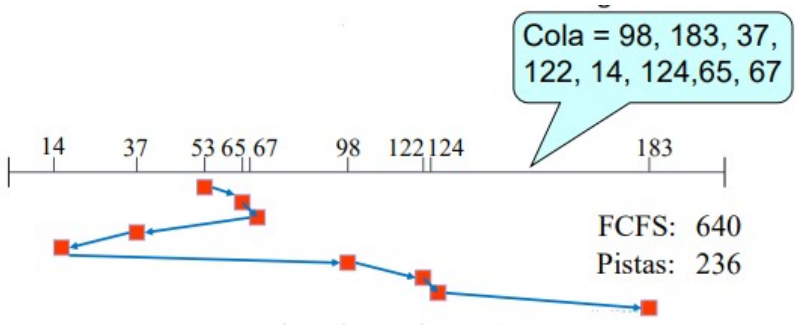


El Tiempo de Búsqueda más Corto Primero SSTF

Parece razonable satisfacer todas las solicitudes que estén próximas a la posición actual del cabezal antes de desplazar el cabezal hasta una posición muy alejada para dar servicio a otras solicitudes. Esta suposición es la base para el algoritmo de búsqueda más corto. Este algoritmo selecciona la solicitud que tenga el tiempo de búsqueda mínimo con respecto a la posición actual del cabezal. Puesto que el tiempo de búsqueda se incrementa a medida que lo hace el número de cilindros recorridos por el cabezal, SSTF selecciona la solicitud pendiente que este más próxima a la posición actual del cabezal.
La planificación SSTF es, esencialmente, un tipo de planificación SJF (shortest-job-first) y, al igual que la planificación SJF, puede provocar la muerte por inanición de algunas solicitudes, por lo cual es inequitativo.

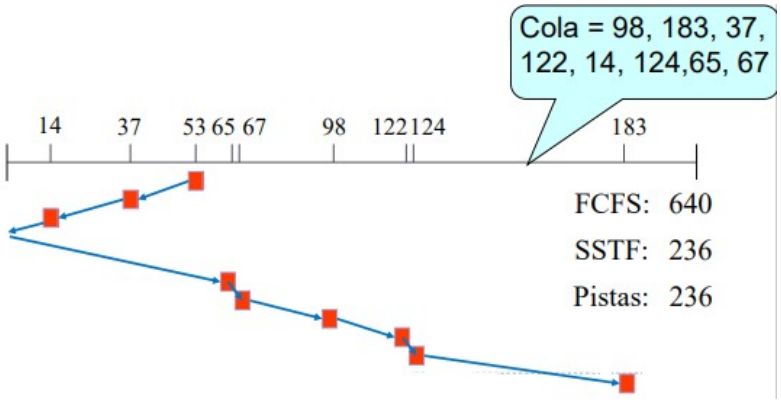
- ✓ El tiempo medio depende de la carga.
- ✓ El tiempo de servicio es menor cuando la cola es más larga.

- ✓ El algoritmo SSTF representa una mejora sustancial con respecto al algoritmo FCFS, no es un algoritmo óptimo.



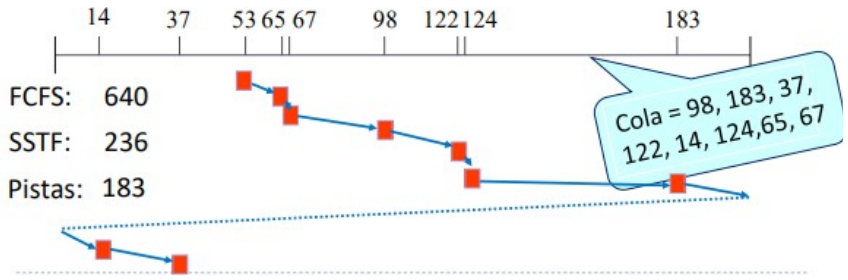
SCAN

En el algoritmo SCAN, o también llamado **algoritmo del ascensor**, el brazo del disco comienza en uno de los extremos del disco y se mueve hacia el otro extremo, dando servicio a las solicitudes a medida que pasa por cada cilindro. En ese otro extremo, se invierte la dirección del cabezal y se continúa dando servicio a las solicitudes. El algoritmo SCAN se denomina también en ocasiones algoritmo del ascensor. Cabe aclarar que por su naturaleza no tiene un tiempo de espera uniforme, ya que si llega una solicitud a la cola y esa solicitud corresponde a un cilindro situado justo delante del cabezal, dicha solicitud será servida casi inmediatamente; por el contrario, una solicitud relativa a un cilindro situado justo detrás del cabezal tendrá que esperar hasta que el brazo alcance el extremo del disco, invierta su dirección y vuelva atrás.



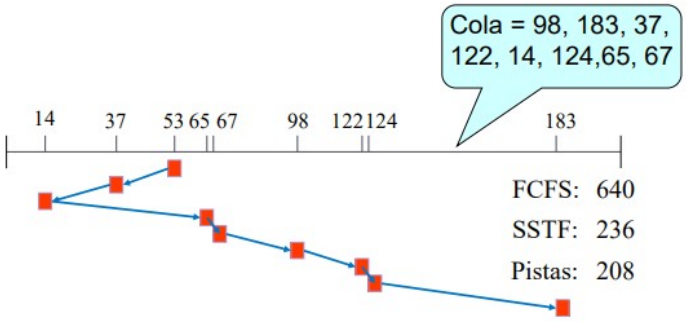
C-SCAN (Circular Scan)

La planificación circular (C-SCAN) es una variante de SCAN diseñada para proporcionar un tiempo de espera más uniforme. Al igual que SCAN, C-SCAN mueve el cabezal de un extremo del disco al otro, prestando servicio a las solicitudes a lo largo de ese trayecto. Sin embargo, cuando el cabezal alcanza el otro extremo, vuelve inmediatamente al principio del disco, sin dar servicio a ninguna solicitud en el viaje de vuelta. El algoritmo de planificación C-SCAN trata esencialmente a los cilindros como si fueran una lista circular que se plegara sobre sí misma conectando el cilindro final con el primer cilindro.



LOOK

Tanto el algoritmo SCAN como el C-SCAN mueven el brazo a través de la anchura completa del disco; sin embargo, en la práctica, ninguna de las versiones del algoritmo SCAN se suele implementar de esta manera. Lo más común es que el brazo sólo vaya hasta el cilindro correspondiente a la solicitud final en cada dirección. Entonces, invierte su dirección inmediatamente, sin llegar hasta el extremo del disco. El brazo va tan lejos en cada dirección como el último requerimiento.



Selección de un Algoritmo de Planificación de Disco

- SSTF resulta bastante común y tiene un atractivo natural, porque mejora la velocidad que puede obtenerse con FCFS.
- SCAN y C-SCAN tienen un mejor comportamiento en aquellos sistemas donde el disco está sometido a una intensa carga, porque es menos probable que provoquen problemas de muerte por inanición.
- Con cualquier algoritmo de planificación, las prestaciones dependen en gran medida del número de solicitudes y del tipo de estas.
- Las solicitudes de servicio de disco pueden verse enormemente influenciadas por el método de asignación de archivos. Un programa que esté leyendo un archivo con asignación contigua generará varias solicitudes que están próximas entre sí en el disco, lo que da como resultado un movimiento de cabezales limitado. Por el contrario, un archivo enlazado o indexado puede incluir bloques que estén muy dispersos por el disco, lo que provoca un movimiento mucho mayor de los cabezales.
- La ubicación de los directorios y de los bloques de índice también es importante. Debido a estas complejidades, el algoritmo de planificación debe escribirse como un módulo independiente del sistema operativo, para poderlo sustituir por otro algoritmo distinto en caso necesario. Tanto SSTF como LOOK son una elección razonable como algoritmo predeterminado.
- Los algoritmos de planificación sólo tienen en cuenta las instancias de búsqueda. Para los discos modernos, la latencia rotacional puede ser casi tan grande como el tiempo medio de búsqueda. Sin embargo, resulta difícil para el sistema operativo realizar una planificación para optimizar la latencia rotacional, porque los discos modernos no revelan la ubicación física de los bloques lógicos. Los fabricantes de disco tratan de aliviar este

problema implementando algoritmos de planificación del disco en el hardware controlador integrado dentro de la unidad de disco.

Administración de Disco

Un disco magnético nuevo es como una pizarra en blanco, se trata simplemente de una placa de material magnético para grabación. Antes de poder almacenar los datos en el disco, es necesario dividir este en sectores que la controladora de disco pueda leer y escribir. Este proceso se denomina **formateo de bajo nivel o formateo físico.** El formateo de bajo nivel llena el disco con una estructura de datos especial para cada sector. La estructura de datos para un sector consta típicamente de una cabecera, un área de datos y una cola. La cabecera y cola contienen información utilizada por la controladora de disco.

Para utilizar un disco para almacenar archivos, el sistema operativo sigue necesitando poder grabar sus propias estructuras de datos en el disco y para ello sigue un proceso en dos pasos:

1. El primer paso consiste en particionar el disco en uno o más grupos de cilindros.
2. Después del particionamiento, el segundo paso es el formato lógico o creación de un sistema de archivos.

Para que una computadora comience a operar debe tener un programa inicial que ejecutar. Este programa inicial de arranque tiende a ser muy simple. Se encarga de inicializar todos los aspectos del sistema, desde los registros de la CPU hasta las controladoras de dispositivo y el contenido de la memoria principal, y luego arranca el sistema operativo. Para llevar a cabo su tarea, el programa de arranque localiza el kernel del sistema operativo en disco, carga dicho kernel en memoria y salta hasta una dirección inicial con el fin de comenzar la ejecución del sistema operativo.

Para la mayoría de las computadoras, el programa de arranque está almacenado en memoria de solo lectura, ROM. Este programa se lo conoce como **bootstrap**. Si bien esta ubicación resulta muy adecuada, porque la ROM no necesita ser inicializada y se encuentra en una dirección fija y no puede verse afectada por virus informáticos, si es necesario cambiar el código deberemos cambiar los chips hardware de la ROM. Por esta razón, la mayoría de los sistemas almacenan un programa cargador de arranque de muy pequeño tamaño en la ROM de arranque, cuya única tarea consiste en cargar un programa de arranque completo en el disco.

Puesto que los discos tienen partes móviles y tolerancias muy pequeñas son bastantes propensos a los fallos. Algunas veces, el fallo que se produce es completo, en cuyo caso será necesario cambiar el disco y restaurar su contenido en el nuevo disco a partir de un soporte de copia de seguridad. Lo más frecuente es que uno o más sectores pasen a ser defectuosos. La mayoría de los discos vienen incluso de fábrica con una serie de bloques defectuosos. Dependiendo del disco y de la controladora que se utilice, estos bloques se gestionan de diversas formas.

Manejo de Espacio de Swapping

La gestión del espacio de intercambio es otra tarea de bajo nivel del sistema operativo. La memoria virtual utiliza el espacio de disco como una extensión de la memoria principal. Puesto que el acceso a disco es mucho más lento que el acceso a memoria, la utilización del espacio de intercambio reduce significativamente las prestaciones del sistema. El objetivo principal del diseño y la implementación del espacio de intercambio es proporcionar la mejor tasa de transferencia posible para el sistema de memoria virtual.

El espacio de intercambio se utiliza de diversas formas en los distintos sistemas operativos, dependiendo de los algoritmos de gestión de memoria que se empleen. Debemos observar que puede resultar más seguro sobrestimar la cantidad de espacio de intercambio requerido, en lugar de subestimarla, porque si un sistema se queda sin espacio de intercambio puede verse

forzado a abortar procesos o puede incluso sufrir un fallo catastrófico. La sobreestimación hace que se desperdicie un espacio de disco que podría, de otro modo, utilizarse para los archivos, pero no provoca ningún daño.
El espacio de intercambio puede residir en uno de dos lugares: puede construirse a partir del sistema de archivos normal o puede residir en una partición de disco separada. Si el espacio de intercambio es simplemente un archivo de gran tamaño dentro del sistema de archivos, pueden usarse las rutinas normales del sistema de archivos para crearlo, nombrarlo y asignarle el espacio. Está técnica, aunque resulta fácil de implementar, también es poco eficiente. Alternativamente, puede crearse un espacio de intercambio en una partición sin formato separada, ya que en este espacio no se almacena ningún sistema de archivos ni estructura de directorio. En lugar de ello, se utiliza un gestor de almacenamiento independiente para el espacio de intercambio, con el fin de asignar y desasignar los bloques de la partición sin formato.

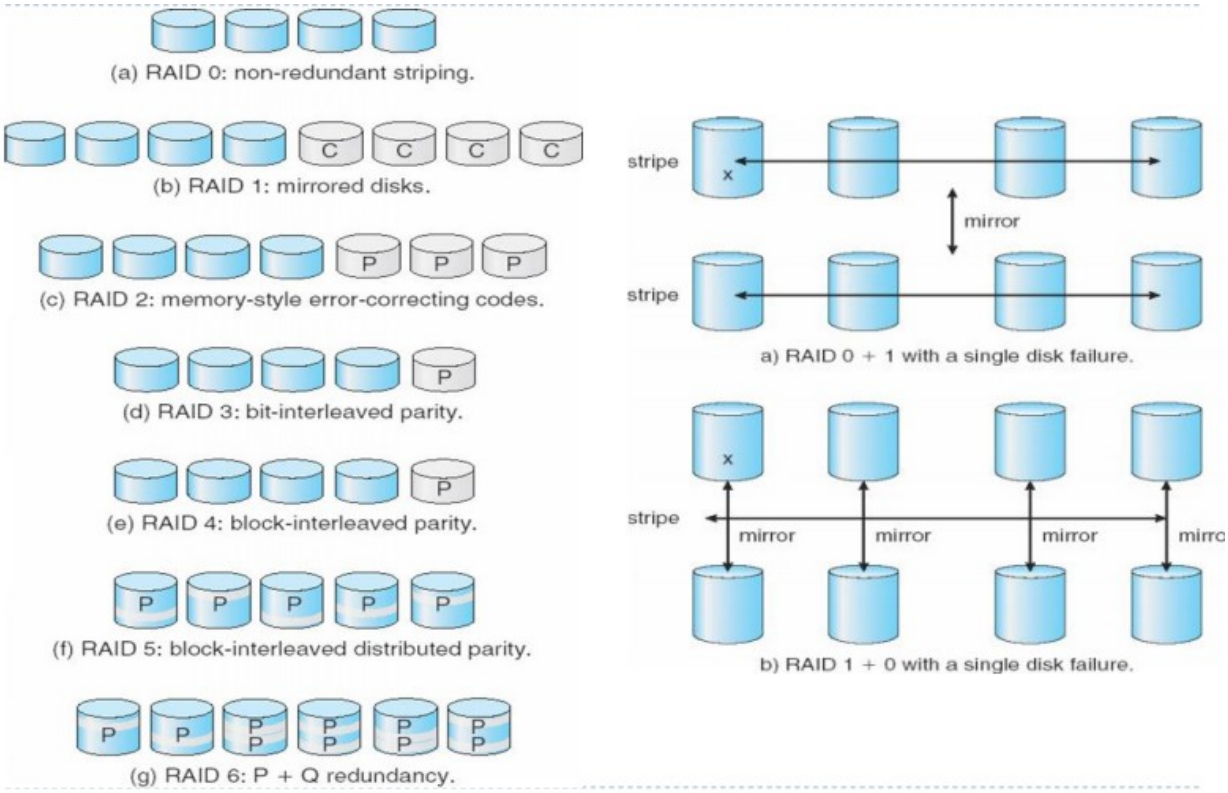
Estructura RAID (Redundant Arrays of Independent Disks)

Múltiples discos proveen confiabilidad vía redundancia. Disponer de un gran número de discos en un sistema permite mejorar la velocidad con la que lo que datos pueden leerse o escribirse, siempre y cuando los discos operen en paralelo. Además, esta configuración también puede permitir mejorar la fiabilidad del almacenamiento de los datos y que puede almacenarse información redundante en ese conjunto de múltiples discos. De este modo, el fallo de uno de los discos no conduce a la pérdida de los datos. Existen diversas técnicas de organización de discos, colectivamente denominadas **matrices redundantes de discos de bajo coste**, que se utilizan comúnmente para resolver estas cuestiones de velocidad y de fiabilidad.
Mejora de la fiabilidad a través de la redundancia: la solución al problema de la fiabilidad consiste en introducir redundancia. Con este sistema, almacenamos información adicional que no es normalmente necesaria pero que puede utilizarse en caso de que falle el disco, para reconstruir la información perdida. Así, incluso si un disco falla, los datos no se pierden.

⇒ La técnica más simple, pero la más cara, para introducir redundancia consiste en duplicar cada uno de los discos. Esta técnica se denomina **duplicación en espejo (mirroring).** Así, cada disco lógico estará compuesto por dos discos físicos y toda escritura se llevará a cabo en ambos discos a la vez. Si uno de los discos falla, pueden leerse los datos del otro disco. Los datos sólo se perderán si el segundo disco fallara antes de que el primero disco estropeado se reemplace. Con este mecanismo, la velocidad a la que pueden gestionarse las solicitudes de lectura se dobla, ya que esas solicitudes pueden enviarse a cualquier de los discos. No obstante, la escritura se reduce ya que debe ser realizada en los dos discos para mantener el espejo.

⇒ También podemos mejorar la velocidad de transferencia dividiendo los datos entre distintos discos. En su forma más simple, el mecanismo de **distribución en bandas de los datos (data striping)** consiste en dividir los bits de cada byte entre múltiples discos; dicha distribución se denomina distribución en bandas de nivel bit. Con este tipo de organización, todos los discos participan en cada acceso (de lectura o escritura) porque el número de accesos que pueden procesarse por segundo es más o menos el mismo que con un único disco, pero cada acceso puede leer más datos en el mismo tiempo que se si utiliza un único disco.

Niveles RAID



La duplicación en espejo proporciona una alta fiabilidad, pero resulta muy cara. La distribución en bandas proporciona una alta velocidad de transferencia de datos, pero no mejora la fiabilidad. Se han propuesto números esquemas para proporcionar redundancia a un menor coste utilizando la idea de distribución en bandas combinada con bits de paridad. Estos esquemas tienen diferentes tipos de compromisos entre el coste y las prestaciones y se clasifican de acuerdo con una serie de niveles denominados niveles RAID.

- **RAID nivel 0:** hace referencia a matrices de discos con distribución de bandas en el nivel de bloques, pero sin ninguna redundancia.
- **RAID nivel 1:** hace referencia a la duplicación en espejo de los discos.
- **RAID nivel 2:** también se conoce como organización con códigos de corrección de errores (ECC) de tipo memoria. Utiliza bit d paridad para detectar ciertos errores. De este modo, el sistema de memoria puede detectar todos los errores de un único bit. Los esquemas de corrección de errores almacenan dos o más bits adicionales u pueden reconstruir los datos si resulta dañado un único bit.
- **RAID nivel 3:** proporciona una mejora con respecto al nivel 2, porque tiene en cuenta que, a diferencia de los sistemas de memoria, las controladoras de disco pueden detectar si un sector ha sido leído correctamente, por lo que puede utilizarse un único bit de paridad tanto para detección como para corrección de errores.
- **RAID nivel 4:** utiliza una distribución en bandas de nivel de bloque, y además mantiene un bloque de paridad en un disco separado, con información de paridad para los bloques correspondientes de los otros N discos. Si uno de los discos falla, puede usarse el bloque de paridad con los bloques correspondientes de los otros discos para restaurar los bloques del disco fallido.
- **RAID nivel 5:** difiere del nivel 4 en que se distribuyen los datos y la información de paridad entre los N+1 discos, en lugar de almacenar los datos en N discos y la información

de paridad en un disco. Para cada bloque, uno de los discos almacena la información de paridad y los otros almacenan los datos.

- **RAID nivel 6:** similar a RAID 5, pero almacena información redundante adicional para proteger los datos frente a múltiples fallos de disco. En lugar de emplearse información de paridad, se utilizan códigos de corrección de errores.
- **RAID nivel 0+1:** hace referencia a una combinación de los niveles de RAID 0 y RAID 1. RAID 0 proporciona la mejora en las prestaciones, mientras que RAID 1 proporciona la fiabilidad. Generalmente, este nivel proporciona un mejor rendimiento que RAID 5. Dobla al número de discos necesarios para el almacenamiento. Se distribuye en bandas un conjunto de discos y luego ese conjunto se duplica en espejo en otro conjunto equivalente.

Implementación de Almacenaje Estable

Los esquemas de bitácora de escritura adelantada requieren almacenaje estable. La información que reside en un almacenamiento estable no se pierde nunca. Para implementar este tipo de almacenamiento, necesitamos replicar la información necesaria en múltiples dispositivos de almacenamiento con modos independientes de fallo. Necesitamos coordinar la escritura de las actualizaciones en forma que se garantice que un fallo durante una actualización no dejará todas las copias en un estado incorrecto y que, cuando nos estemos recuperando de un fallo, podamos forzar a todas las copias para que adopten un estado coherente y correcto, incluso si se produce otro fallo durante la recuperación.

Dispositivos de Almacenaje Terciarios

En una computadora puede reducirse el coste global utilizando muchos discos de bajo coste con una misma unidad. El bajo coste es una característica definitoria del almacenamiento terciario. Puesto que el coste es tan importante, en la práctica el almacenamiento terciario se construye con soportes extraíbles. Los ejemplos más comunes son los disquetes, las cintas y los CD, DVD y los pendrives.

Aspectos del Sistema Operativo

Dos de los principales cometidos de un sistema operativo son gestionar los dispositivos físicos y presentar una abstracción de máquina virtual a las aplicaciones. Para los discos duros el SO provee dos abstracciones:

1. Una es el dispositivo crudo o sin formato, compuesto simple por una matriz de bloques de datos.
2. La otra es un sistema de archivos. Para el sistema de archivos en un disco magnético, el sistema operativo pone en cola y planifica las solicitudes entrelazadas emitidas por varias aplicaciones.

La mayoría de los sistemas operativos pueden gestionar los discos extraíbles de forma casi exactamente igual que los discos fijos. Cuando se inserta un cartucho en blanco en una unidad, el cartucho debe formatearse, con lo que se escribe en el disco un sistema de archivos vacío. Este sistema de archivos se utiliza igual que los sistemas de archivo residentes en el disco duro.
Las cintas suelen manejarse de modo distinto. El sistema operativo, usualmente, presenta las cintas como soportes de almacenamiento sin formato. Las aplicaciones no abren un archivo en la cinta, sino que abren la unidad de cinta completa como un dispositivo sin formato. Usualmente, la unidad de cinta queda entonces reservada para el uso exclusivo por parte de dicha aplicaciones, hasta que la aplicación termina o hasta que cierra el dispositivo de cinta.
Resulta fácil ver los problemas que pueden surgir debido a esta forma de utilizar las cintas. Puesto que cada aplicación tiene sus propias reglas en cuanto al modo de organizar una cinta, cada cinta llena de datos sólo puede, por regla general, ser utilizada por el programa que la creó.

Otra cuestión que el sistema operativo necesita resolver es cómo nombrar los archivos en los soportes extraíbles.

Cuestiones de rendimiento

Al igual que sucede con cualquier otro componente del sistema operativo, los tres aspectos más importantes del rendimiento del almacenamiento terciario son la velocidad, la fiabilidad y el coste.

- **Velocidad**: la velocidad del almacenamiento terciario puede analizarse desde dos puntos de vista: ancho de banda y latencia. El ancho de banda se mide en bytes por segundo; el ancho de banda sostenido es la velocidad media de transferencia de datos durante una trasferencia de gran envergadura. El ancho de banda efectivo mide el promedio para toda la duración de la operación de E/S.
- **Confiabilidad**: si tratamos de leer ciertos datos y no podemos hacerlo debido a un fallo de la unidad o del soporte físico, desde el punto de vista práctico el tiempo de acceso será infinitamente largo y el ancho de banda será infinitamente pequeño. Es importante comprender cuál es el grado de fiabilidad de los soportes extraíbles.
- **Costo**: el costo de los almacenamientos es inversamente proporcional al tamaño.

Máquinas virtuales

Virtualizar

Virtual

- Dícese de lo que tiene virtud para producir un efecto, aunque no lo produce de presente. (diccionario).
- Que no tiene existencia real sino aparente (diccionario)

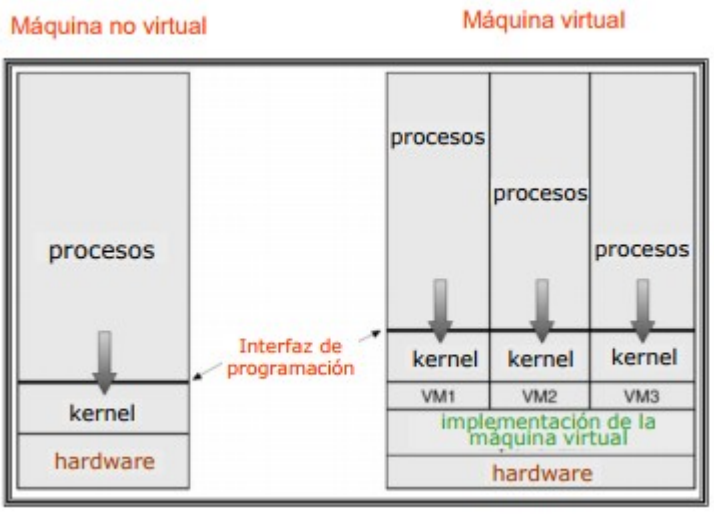
¿Por qué virtualizar?

- Reduce el costo e incrementa la eficiencia de los existentes recursos de hardware.
- Lograr más en menos tiempo
  - Ejecute varios sistemas operativos en una sola Computadora
  - Reduzca el número de computadores físicos que se requieren
- Facilitar la migración de aplicaciones
- Agilizar la implementación
  - Pruebe nuevo software y sistemas operativos antes de su implementación
- Acelerar el desarrollo de aplicaciones
  - Incremente el aseguramiento de calidad al probar en diferentes sistemas operativos utilizando máquinas virtuales
  - Reduzca el tiempo para salir al mercado con menos reconfiguración

Maquina virtual

La idea central de una máquina virtual es la abstracción del hardware de una computadora en varios ambientes de ejecución diferentes, creando la ilusión de que cada ambiente de ejecución está corriendo en su propia computadora privada.

- Una máquina virtual provee una interfaz idéntica al hardware primitivo subyacente.



- El sistema operativo crea la ilusión de múltiples procesos, cada uno ejecutando en su propio procesador con su propia memoria (virtual).
- Cada invitado es provisto con una copia (virtual) de la computadora

Historias y beneficios de las máquinas virtuales

⇒ Aparecieron comercialmente en las mainframes de IBM en 1972.
⇒ Fundamentalmente, múltiples ambientes de ejecución (diferentes Sos) pueden compartir el mismo hardware. Están protegidos uno de otro. Puede permitirse, en forma controlada, compartir archivos. Conmuta uno con otros sistemas físicos vía red.
⇒ Útil para desarrollo, testing.
⇒ "Open Virtual Machine Format", un formato standard de máquinas virtuales, permite a una VM correr dentro de diferentes plataformas (host) de máquinas virtuales

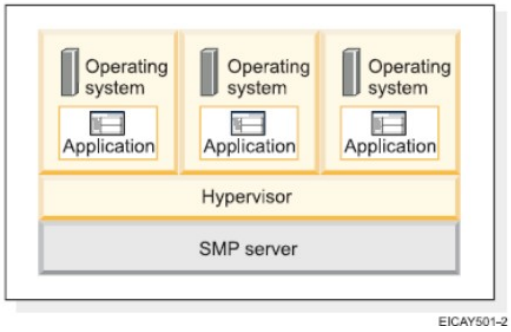
Características y clasificación

✓ Misma arquitectura distintos Sistemas Operativos
✓ Diferentes arquitecturas.

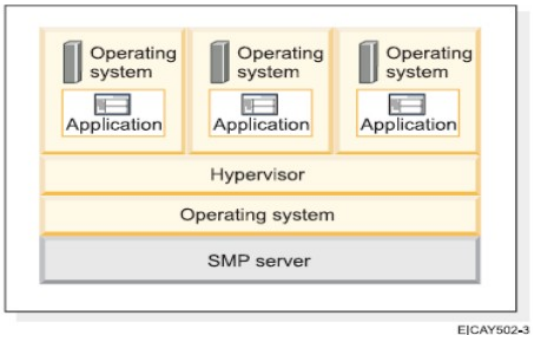
Clasificación

- Tipo 0 – son soluciones basados en hardware, que proveen soporte para la creación y administración vía el firmware.
- Tipo 1 – Hypervisors ejecutan directamente sobre el hardware de la máquina.
- Tipo 2 – Hypervisors ejecutan sobre el sistema operativo host que provee los servicios de virtualización.

Tipo 1



Tipo 2



Paravirtualización

Una técnica en la cual el Sistema operativo invitado es modificado para trabajar en cooperación con el VMM (Virtual machine manager) para optimizar el rendimiento. Los invitados deben ser modificados para correr en un hardware paravirtualizado.
Ventaja: Alcanzar mayor eficiencia en el uso de los recursos y una capa de virtualización de menor tamaño.

Protección y Seguridad - Módulo 13

Protección

Los procesos en un sistema operativo deben protegerse de las actividades realizadas por otros procesos. Para proporcionar dicha protección, podemos utilizar diversos mecanismos para

garantizar que solo los procesos que hayan obtenido la autorización adecuada del sistema operativo puedan operar sobre los archivos, los segmentos de memoria, sobre la CPU y sobre otros recursos del sistema.
El concepto de protección hace referencia a un mecanismo para controlar el acceso de los programas, de los procesos o de los usuarios a los recursos definidos por el sistema informático. Este mecanismo debe proporcionar un medio de especificar los controles que hay que imponer, junto con un modo de imponerlos. Podemos distinguir entre los conceptos protección y seguridad; este último es una medida de la confianza en que se puedan preservar la integridad de un sistema y sus datos.

Objetivos de la protección

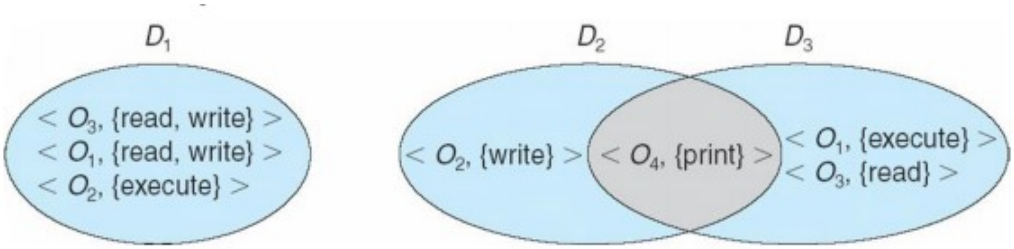
- El SO consiste de una colección de objetos, hardware o software.
- Cada objeto tiene un único nombre y puede ser accedido por un conjunto de operaciones bien definidas.
- El **problema de protección** asegura que cada objeto es accedido correctamente y solo por aquellos procesos que les está permitido ha

Principios de Protección

Podemos utilizar un principio director a lo largo del diseño de un sistema operativo. Ajustarnos a este principio simplifica las decisiones de diseño y hace que el sistema continúe siendo coherente y fácil de comprender. Uno de los principios directores clave y que ha resistido al paso del tiempo a la hora de proporcionar protección es el **principio del mínimo privilegio.** Este principio dicta que a los programas, a los usuarios, incluso a los sistemas se les concedan únicamente los suficientes privilegios para llevar a cabo sus tareas.
Un sistema operativo que se ajuste al principio del mínimo privilegio implementará sus características, programas, llamadas al sistema y estructuras de datos de modo que el fallo o el compromiso de un componente provoquen un daño mínimo y no permitan realizar más que un daño mínimo.

Estructura de Dominios

Un sistema informático es una colección de procesos y objetos. Cada objeto tiene un único nombre y puede ser accedido por un conjunto de operaciones bien definidas. Los objetos son tipos de datos abstractos. Las operaciones posibles pueden depender de cada objeto.
A un proceso solo se le debe permitir acceder a aquellos recursos para los que tenga autorización. Además, en cualquier instante determinado, un proceso solo debería poder acceder a aquellos recursos que necesite actualmente para completar su tarea.
El problema de la protección asegura que cada objeto es accedido correctamente y solo por aquellos procesos que les está permitido hacerlo.
Para facilitar este esquema, un proceso opera dentro de un dominio de protección, que especifica los recursos a los que el proceso puede acceder. Cada dominio define un conjunto de objetos y los tipos de operaciones que pueden invocarse sobre cada objeto. La capacidad de ejecutar una operación sobre un objeto es un derecho de acceso. Un dominio es una colección de derechos de acceso, cada uno de los cuales es una pareja ordenada <nombre-objeto, conjunto-derechos>, donde el conjunto de derechos es un subconjunto de todas las operaciones válidas que pueden ser realizadas por el objeto.
Los dominios no tienen por qué ser disjuntos, sino que pueden compartir derechos de acceso.



Implementación de Dominios (UNIX)

En el sistema operativo UNIX, un dominio está asociado con el usuario. La conmutación de dominio se corresponde con un cambio temporal en la identificación del usuario. Este cambio se lleva a cabo a través del sistema de archivos de la forma siguiente: con cada archivo hay asociada una identificación de propietario y un bit de dominio (conocido como el bit setuid); cuando el bit setuid está activado y un usuario ejecuta dicho archivo, el ID de usuario se configura con el valor correspondiente al propietario del archivo; sin embargo, cuando el bit está desactivado, el ID del usuario no se modifica. Cuando se completa la ejecución la identificación de usuario es retorna a su original.

Debe tenerse un gran cuidado a la hora de escribir los programas privilegiados. Cualquier descuido puede provocar una total falta de protección en el sistema. Generalmente, estos programas son los primeros en ser atacados por aquellas personas que tratan de irrumpir dentro de un sistema.

Matriz de Acceso

El modelo de protección basado en dominios puede contemplarse de forma abstracta como una matriz, denominada matriz de acceso. Las filas de la matriz de acceso representan dominios y las columnas representan objetos. Cada entrada de la matriz está compuesta de un conjunto de derechos de acceso. La entrada *acceso(i,j)* define el conjunto de operaciones que un proceso que se ejecute en el dominio D_i puede invocar sobre el objeto O_j.

Si un proceso en Dominio D_i trata de hacer “op” sobre el objeto O_j, entonces “Op” debe estar en la matriz de acceso.

La matriz de acceso proporciona el mecanismo apropiado para definir e implementar un control estricto de la asociación tanto estática como dinámica entre procesos y dominios. Cuando conmutamos un proceso de un dominio a otro, ejecutamos una operación (switch) sobre un objeto (el dominio). Podemos controlar la conmutación de dominios incluyendo los dominios entre los objetos de la matriz de acceso. De forma similar, cuando cambiamos el contenido de la matriz de acceso, realizamos una operación sobre un objeto: la propia matriz de acceso. Entonces debemos considerar a cada entrada de la matriz de acceso como un objeto que hay que proteger. Ahora, sólo necesitamos considerar las operaciones que sean posibles sobre estos objetos y decidir cómo queremos que los procesos puedan ejecutar dichas operaciones.

Estas operaciones sobre los dominios y la matriz de acceso no son por sí mismas importantes, pero ilustran la capacidad del modelo de la matriz de acceso para permitir la implementación y control de requisitos de protección dinámicos. Pueden crearse dinámicamente nuevos objetos y nuevos dominios e incluirlos en el modelo de la matriz de acceso. Sin embargo, sólo hemos mostrado que el mecanismo básico existe; los diseñadores del sistema y los usuarios deben tomar las decisiones de políticas relativas a qué dominios deben poder acceder a qué objetos y en qué manera.

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

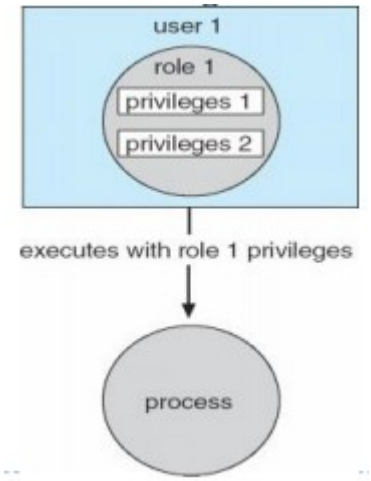
Implementación de la Matriz de Acceso

- **Tabla global:** La implementación más simple de la matriz de acceso es una tabla global compuesta de un conjunto de tripletas ordenadas <dominio, objeto, conjunto-derechos>. La tabla es usualmente muy grande y no puede, por tanto, ser conservada en memoria principal, por lo que hacen falta operaciones adicionales de E/S. A menudo se utilizan técnicas de memoria virtual para gestionar esta tabla. Además, resulta difícil aprovechar las agrupaciones especiales de objetos o dominios.
- **Lista de acceso para los objetos:** Cada columna de la matriz de acceso puede implementarse como una lista de acceso de un objeto. Obviamente, las entradas vacías pueden descartarse. La lista resultante para cada objeto estará compuesta por una serie de parejas ordenadas <dominio, conjunto-derechos>, que definen todos los dominios que tenga un conjunto de derechos de acceso no vacío para dicho objeto.
  Define quien puede realizar que operación →
  - Dominio 1 = Read, Write
  - Dominio 2 = Read
  - Dominio 3 = Read
- **Listas de las capacidades para los dominios:** En lugar de asociar las columnas de la matriz de acceso con los objetos en forma de listas de acceso, podemos asociar cada fila con su dominio. Una lista de capacidades para un dominio es una lista de objetos junto con las operaciones permitidas sobre esos objetos. Cada objeto se suele representar mediante su dirección o nombre físico, denominada capacidad. La simple posesión de la capacidad significa que el acceso está permitido.
  Para cada dominio que operaciones están permitidas sobre que objetos →
  - Objeto 1 – Read
  - Objeto 4 – Read, Write, Execute
  - Objeto 5 – Read, Write, Delete, Copy

Control de Accesos

La protección puede ser aplicada también a recursos físicos.
Solaris 10 amplía el sistema de protección disponible en el sistema operativo añadiendo explícitamente el principio del mínimo privilegio mediante el control de acceso basado en roles. Esta funcionalidad gira en torno a los privilegios. Un privilegio es el derecho a ejecutar una llamada al sistema o usar una opción dentro de dicha llamada al sistema. Podemos asignar privilegios a los procesos, limitando éstos a exactamente el tipo de acceso que necesitan para llevar a cabo su tarea. Los privilegios y programas también pueden asignarse a roles. A los

usuarios se les asignan roles basándose en contraseñas asignadas a los roles. De esta forma, un usuario puede adoptar un rol que activa un privilegio, permitiendo al usuario ejecutar un programa para llevar a cabo una tarea específica. Esta implementación de los privilegios reduce el riesgo de seguridad asociado al setuid.



Revocación de Derechos de Acceso

En un sistema de protección dinámico, puede que necesitamos en ocasiones revocar derechos de acceso a objetos compartidos por diferentes usuarios.

Con el esquema basado en lista de acceso, la revocación resulta sencilla. Se analiza la lista de acceso en busca de los derechos de acceso que hay que revocar y se los borra de la lista. La revocación es inmediata y puede ser general o selectiva, total o parcial y permanente o temporal.

Las capacidades, sin embargo, presentan un problema mucho más difícil de cara a la revocación. Puesto que las capacidades están distribuidas por todo el sistema, debemos encontrarlas antes de poder revocarlas. Entre los esquemas que pueden utilizarse para implementar la revocación en un sistema basado en capacidades podemos citar:

- Readquisición: Periódicamente, se borran las capacidades de cada dominio. Si un proceso quiere usar una capacidad, puede que se encuentre con que esa capacidad ya ha sido borrada. El proceso puede entonces tratar de volver a adquirir la capacidad. Si se ha revocado el acceso, el proceso no será capaz de efectuar esa adquisición.
- Retropunteros: Con cada objeto se mantiene una lista de punteros, que hace referencia a todas las capacidades asociadas con ese objeto. Cuando se necesita realizar una revocación, podemos seguir esos punteros, cambiando las capacidades según sea necesario.
- Indirección: Las capacidades apuntan indirectamente, en lugar de directamente, a los objetos. Cada capacidad apunta a una entrada unívoca dentro de una tabla global, que a su vez apunta al objeto. Implementamos la revocación analizando la tabla global en busca de la entrada deseada y borrándola. Entonces, cuando se intenta realizar un acceso, se verá que la capacidad está apuntado a una entrada ilegal de la tabla.
- Claves: Una clave es un patrón distintivo de bits que pueden asociarse con una capacidad. Esta clave define en el momento de crear la capacidad y no puede ser nunca modificada ni inspeccionada por el proceso que posee la capacidad. Con cada objeto hay asociada una clave maestra. El proceso de revocación sustituye la clave maestra con un nuevo valor, invalidando las capacidades anteriores del objeto.

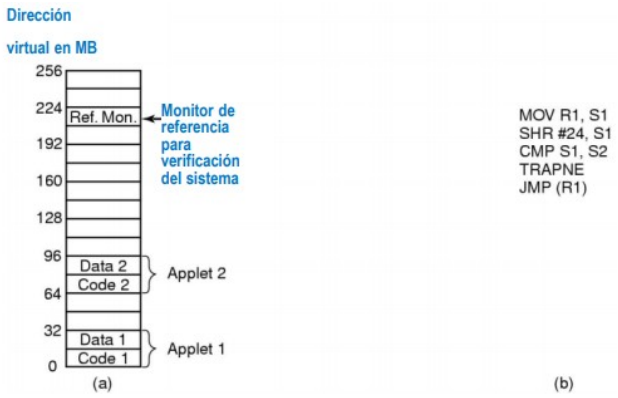
Protección Basada en Lenguajes

Las políticas de uso de los recursos pueden variar, dependiendo de la aplicación, y pueden estar sujetas a cambios a lo largo del tiempo. Por estas razones, la protección ya no puede ser considerada exclusivamente como una preocupación del diseñador del sistema operativo, sino que también debe estar disponible como herramienta para que la use el diseñador de aplicaciones, de modo que los recursos de un subsistema de aplicación puedan estar protegidos frente a las modificaciones o frente a la influencia de errores.

Es aquí donde entran dentro del panorama los lenguajes de programación. La especificación de protección en lenguajes de programación permite una descripción en alto nivel de políticas para la alocación y uso de recursos. La implementación del lenguaje puede forzar software para protección cuando la verificación automática soportada por hardware no está disponible.

Código Móvil

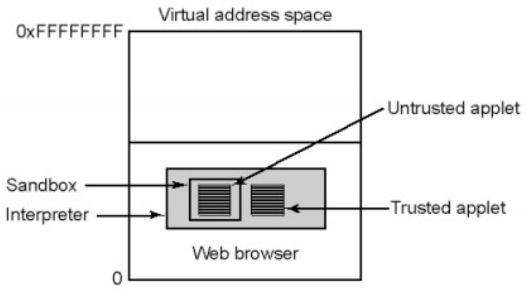
Una técnica para protegerse de las amenazas, es ejecutar un programa en lo que se denomina cajón de arena, que es una sección emulada o controlada del sistema. El software antivirus analiza el comportamiento del código en el cajón de arena antes de dejar que se ejecute sin monitorización.



(a) Memoria dividida en cajas de arena de 1-MB

(b) Una forma de verificar la validez de una instrucción

Código móvil (1) – "Cajas de arena"



Los applets pueden ser interpretados por el browser de Web

Código móvil (2)

Seguridad – Modulo 15

Introducción

La seguridad requiere no solo un adecuado sistema de protección, sino también tener en cuenta el entorno externo dentro del que opera el sistema. Un sistema de protección no será efectivo si

la autenticación de los usuarios se ve comprometida o si un programa es ejecutada por un usuario no autorizado.

El problema de Seguridad

En muchas aplicaciones, merece la pena dedicar un esfuerzo considerable a garantizar la seguridad de un sistema informático, tales como los sistemas comerciales de gran envergadura que contenga datos financieros o relativos a corporaciones.
Decimos que un sistema es seguro si sus recursos se utilizan de la forma prevista y si se accede a ellos tal como se pretendía, en todas las circunstancias. Desafortunadamente, no es posible conseguir una seguridad total; a pesar de ello, debemos prever mecanismos para hacer que los fallos de seguridad constituyan la excepción, en lugar de la norma.
Un intruso o cracker es aquella persona que intenta romper la seguridad. Una **amenaza** es potencialmente una violación a la seguridad. Un **ataque** es un intento de romper la seguridad. Este puede ser accidental o malicioso. Es más fácil protegerse contra un uso accidental que contra uno malicioso ya que principalmente los mecanismos de protección forman la base de defensa frente a posibles accidentes.

Violaciones de Seguridad

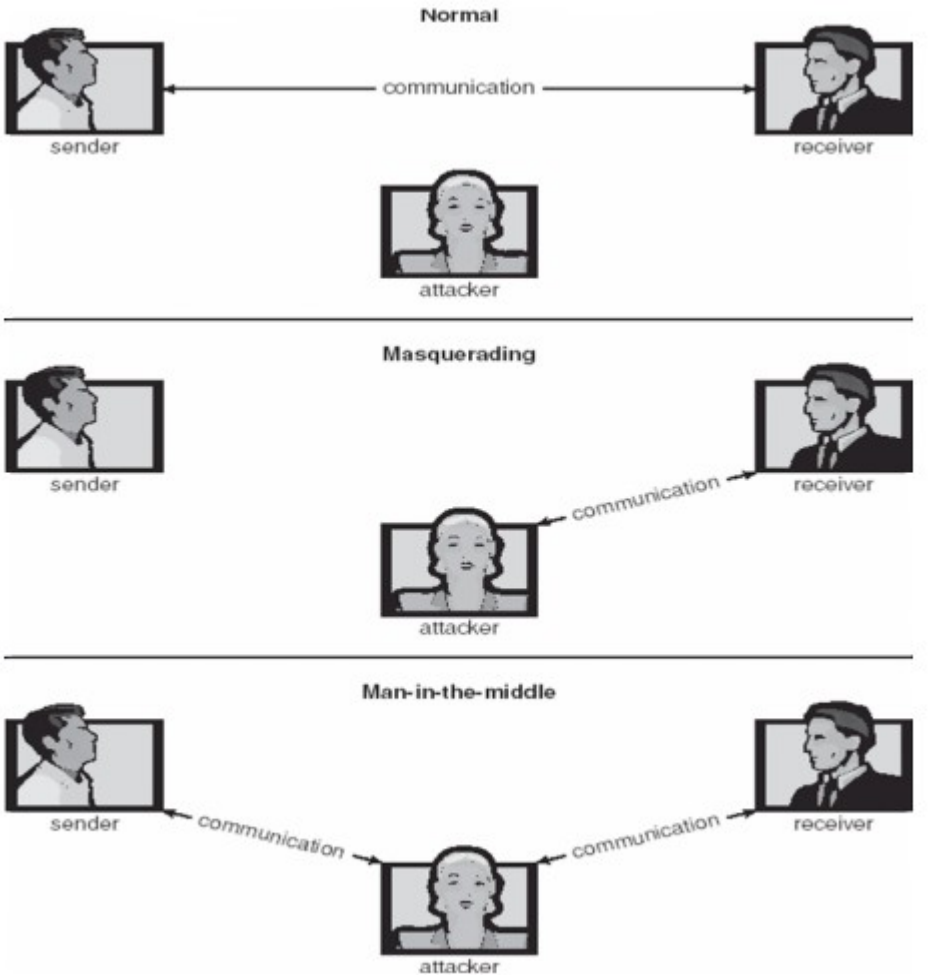
Las violaciones de seguridad o la mala utilización de un sistema pueden clasificarse en dos categorías: intencionadas o accidentales. Resulta más fácil protegerse frente a una mala utilización accidental que frente a otra maliciosa. Principalmente, los mecanismos de protección forman la base de la defensa frente a posibles accidentes.
Existen diferentes formas de violaciones de seguridad:

- ⇒ **Fallo de confidencialidad**: Este tipo de violación implica la lectura no autorizada de determinados datos privados. La captura de datos secretos en un sistema o en un flujo de datos puede reportar al intruso un beneficio monetario directo.
- ⇒ **Fallo de integridad:** Este tipo de ataque implica la modificación no autorizada de los datos. Estos ataquen pueden provocar que se atribuya una cierta responsabilidad a alguien que es inocente o que se modifique el código fuente de una aplicación comercial de cierta importancia.
- ⇒ **Fallo de disponibilidad:** implica la destrucción no autorizada de datos. Algunos atacantes prefieren causar daño y adquirir un cierto renombre en lugar de obtener beneficios financieros. Ejemplo: sustitución de la página de entrada de un sitio web.
- ⇒ **Robo de servicio:** Este tipo de violación de seguridad implica el uso no autorizado de recursos. Ejemplo: un intruso puede instalar un demonio en un sistema que actué como servidor de archivos.
- ⇒ **Negación de Servicio (Denial of service)** Esta violación implica impedir el uso legítimo del sistema. Los ataques de denegación de servicio son en ocasiones accidentes. Ejemplo: un programa gusano que ataco internet hace ya unos cuantos años y que se transformó en un ataque DOS cuando un error en el propio programa hizo que no se retardara su rápida diseminación.

Los atacantes utilizan diversos métodos estándar en sus intentos de romper la seguridad de los sistemas:

- ❖ Mascarada: es el más común, en la que un participante en una comunicación pretende ser otra persona. Mediante la mascarada, los atacantes rompen la autenticación, es decir, la corrección de la identificación; como consecuencia, pueden obtener tipos de acceso que normalmente no les estarían permitidos o escalar sus privilegios, es decir, obtener privilegios que normalmente no podrían disfrutar.

- Ataque replay: es otro ataque común que consiste en reproducir un intercambio de datos previamente capturado: los ataques de reproducción consisten en la repetición maliciosa o fraudulenta de una transmisión de datos válida. En ocasiones, esa reproducción constituye el propio ataque, como por ejemplo cuando se repite una solicitud para transferir dinero, pero frecuentemente la reproducción se realiza en conjunción con una modificación del mensaje.
- Ataque por interposición (man-in-the-midle): un atacante se introduce dentro del flujo de datos de una comunicación, haciéndose pasar por el emisor a ojos del receptor y viceversa.
- Sesión de toma de control: En una comunicación por red, un ataque de imposición puede estar precedido de un secuestro de sesión, en el que se intercepta una sesión de comunicación activa.



Niveles de Medidas de Seguridad

Para proteger un sistema, debemos adoptar las necesarias medidas de seguridad en cuatro niveles distintos:

1. **Físico**: El nodo o nodos que contengan los sistemas informáticos deben dotarse de medidas de seguridad físicas frente a posibles intrusiones armadas por parte de los potenciales intrusos.
2. **Humano**: La autorización de los usuarios debe llevarse a cabo con cuidado, para garantizar que sólo los usuarios apropiados tenga acceso al sistema. Sin embargo, incluso los usuarios autorizados pueden verse “motivados” para permitir que otros usen su acceso. También pueden ser engañados para permitir el acceso a otros, mediante técnicas de ingeniería social. Uno de los tipos de ataque basado en las técnicas de ingeniería social es denominado

phishing; con este tipo de ataque, un correo electrónico o página web de aspecto autentico llevan a engaño a un usuario que introduzca información confidencial. Otra técnica comúnmente utilizada es el análisis de desperdicios, un término general para describir el intento de recopilar información para poder obtener un acceso no autorizado a la computadora. Estos problemas de seguridad son cuestiones relacionadas con la gestión y con el personal, más que problemas relativos a los sistemas operativos.

3. **Sistemas Operativos**: El sistema debe autoprotegerse frente a los posibles fallos de seguridad accidentales o premeditados. Un proceso que este fuera de control podría llegar a constituir un ataque accidental de denegación de archivo
4. **Red**: Son muchos los datos en los modernos sistemas informáticos que viajan a través de líneas arrendadas privadas, de líneas compartidas como Internet, de conexiones inalámbricas o de líneas de acceso telefónico. La intercepción de estos datos podría ser tan dañina como el acceso a una computadora.

Si queremos garantizar la seguridad del sistema operativo, es necesario garantizar la seguridad en los primeros dos niveles. Cualquier debilidad en uno de los niveles altos de seguridad (físico o humano) permitirá puentear las medidas de seguridad que son estrictamente de bajo nivel. Así, la frase que afirma que una cadena es tan fuerte como el más débil de los eslabones es especialmente cierta cuando hablamos de la seguridad de los sistemas. Para poder mantener la seguridad, debemos contemplar todos los aspectos.

👁 El sistema debe proporcionar mecanismos de protección para permitir la implementación de las características de seguridad. Sin la capacidad de autorizar a los usuarios y procesos, de controlar su acceso y de registrar sus actividades, seria imposible que un sistema operativo implementara medidas de seguridad o se ejecutara de forma segura.

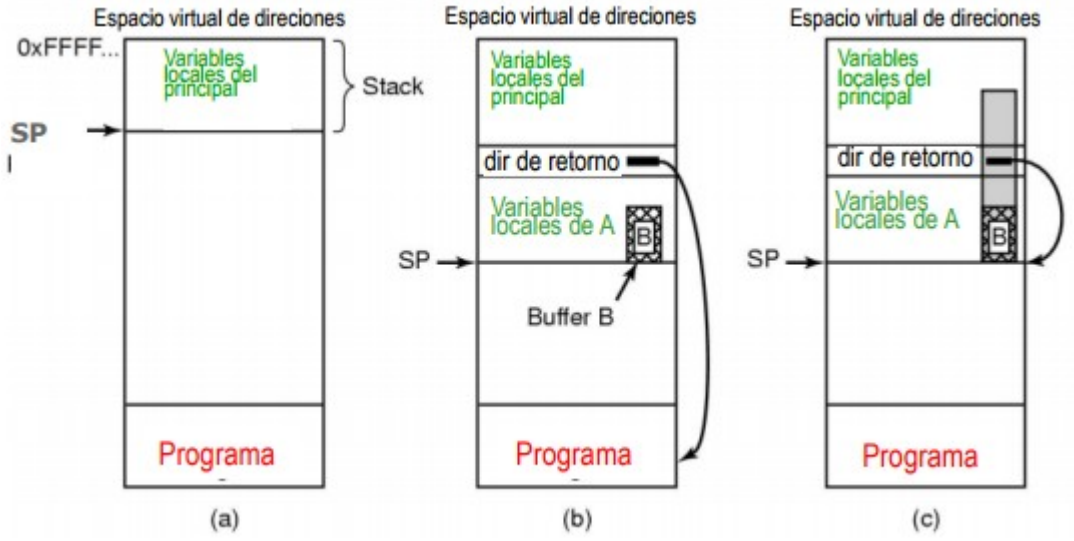
Programas Peligrosos

Los procesos son, junto con el kernel, el único medio de realizar trabajo útil en una computadora. Por tanto, un objetivo común de los piratas informáticos consiste en escribir un programa que cree una brecha de seguridad o que haga que un proceso normal cambie su comportamiento y cree esa brecha de seguridad.

- **Caballo de Troya:** Muchos sistemas tienen mecanismos para permitir que programas escritos por unos usuarios sean ejecutados por otros. Si estos programas se ejecutan en un dominio que proporcione los derechos de acceso del usuario ejecutante, los otros usuarios podrían utilizar inapropiadamente estos derechos. Un segmento de código que utilice inapropiadamente su entorno se denomina caballo de Troya.
  Otra variante del caballo de Troya es el spyware. Los programas spyware acompañan en ocasiones a ciertos programas que el usuario haya decido instalar. En la mayoría de los casos, se incluyen con programas tipo freeware o shareware, pero en otras ocasiones también se incluyen en software de carácter comercial. El objetivo del spyware es descargar anuncios para mostrarlos en el sistema del usuario, crear ventajas de explorador emergentes cuando se visiten sitios o capturar información del sistema del usuario y enviarla a un sitio central. Este último modo es un ejemplo de una categoría general de ataques conocida como canales encubiertos, a través de los cuales tienen lugar comunicaciones subrepticias.
- **Puerta trasera o puerta trampa**: El diseñador de un programa o sistema puede dejar detrás suyo un agujero en el software que sólo él sea capaz de utilizar. Por ejemplo, el código puede tratar de detectar un ID de usuario o una contraseña específicos y, al detectarlo, evitar los procedimientos de seguridad normales. Podría incluirse una puerta trasera inteligente dentro del propio compilador. Las puertas traseras representan un problema porque para detectarlas, tenemos que analizar todo el código fuente de todos los

componentes de un sistema, una tarea muy extensa considerando la cantidad de líneas de código de los sistemas de software.

- **Bomba Lógica:** Considere un programa capaz de iniciar un incidente de seguridad sólo cuando se dan determinadas circunstancias. Sería difícil de detectar porque, en condiciones normales de operación, no existiría ningún agujero de seguridad. Sin embargo, cuando se satisficiera un conjunto predefinido de parámetros, se crearía el agujero de seguridad. Este escenario se conoce con el nombre de bomba lógica. Un programador, por ejemplo, podría escribir código para detectar si todavía continúa contratado por la empresa; en caso de que dicha comprobación fallara, podría crearse un demonio para permitir el acceso remoto, o podría iniciarse un determinado fragmento de código que provocara algún tipo de daño en la instalación.
- **Rebalse de Stack y Buffer:** El ataque por desbordamiento de pila o de búfer es la forma más común para que un atacante externo al sistema a través de una conexión de red obtenga acceso no autorizado al sistema operativo. Esencialmente, lo que el ataque hace es aprovechar un error de un programa. El resultado de la ejecución del código de este programa de ataque será una shell raíz o la ejecución de otro comando privilegiado.

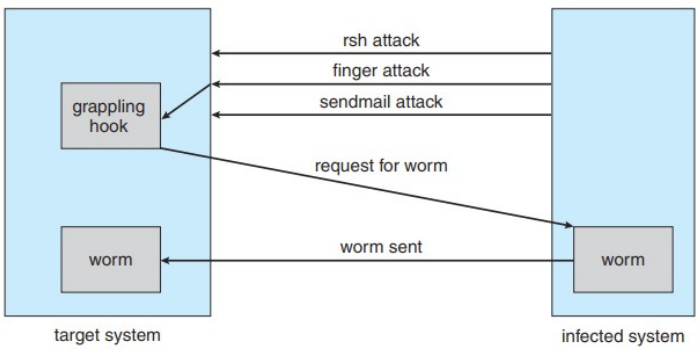


(a) Situación cuando el programa principal está corriendo
(b) Luego del llamado al programa A
(c) El rebalse de buffer mostrado en gris

Amenazas al Sistema y Red

Las amenazas del sistema y de la red implica el abuso de los servicios y las conexiones de red. Las amenazas del sistema y de la red crean una situación en la que se utilizan inapropiadamente los recursos del sistema operativo y los archivos del usuario.

- **Gusanos**: Un gusano es un proceso que utiliza un mecanismo de reproducción para afectar al rendimiento del sistema. El gusano crea copias de sí mismo, utilizando recursos del sistema y en ocasiones impidiendo operar a todos los demás procesos. En las redes informáticas, los gusanos son particularmente potentes, ya que pueden reproducirse de un sistema a otro y colapsar una red completa. El Worm Internet explotaba características de red UNIX y bugs en los programas finger y sendmail.



- **Barrido de Pórticos**: El escaneo de puertos no es un ataque, sino más bien un método para que los piratas informáticos detecten las vulnerabilidades del sistema que puedan ser atacadas. El escaneo de puertos se realiza normalmente de forma automatizada, lo que implica utilizar una herramienta que trate de crear una conexión TCP/IP a un puerto o rango de puertos específicos.
- **Denegación de servicio**: Los ataques DOS están dirigidos no a obtener información o robar recursos, sino a impedir el uso legítimo de un sistema o funcionalidad. La mayoría de los ataques de denegación de servicio afectan a sistemas en los que el atacante no ha penetrado. De hecho, lanzar un ataque que impida el uso legítimo de un sistema resulta más sencillo que irrumpir en una maquina o instalación.
  Este tipo de ataques se realizan generalmente a través de la red. Se los puede clasificar en dos categorías:
  1. El primer caso es el de los ataques que consumen tantos recursos de la máquina atacada que prácticamente no pueden realizarse con ella ningún trabajo útil. Por ejemplo, al hacer clic en un sitio web podría descargarse una applet Java que consumiera todo el tiempo de CPU disponible o se dedicara a abrir una serie infinita de ventanas emergentes.
  2. El segundo caso de ataque implica hacer caer la red o la instalación. Este tipo de ataque es el resultado de un abuso de alguna de las funciones fundamentales de TCP/IP. Generalmente, es imposible prevenir los ataques de denegación de servicio ya que utilizan mismos mecanismos que la operación normal.
- **Denegación de servici**o: son todavía es más difícil de prevenir y de solucionar. Estos ataques se inician desde múltiples sitios a la vez, dirigidos hacia un objeto común. Sobrecarga de la computadora blanco evitando que haga algún trabajo útil.

En ocasiones, un determinado sitio puede no ser ni siquiera consciente de que está siendo atacado. Puede resultar difícil determinar si la ralentización de un sistema se debe simplemente a un pico de utilización del mismo o a un ataque.

Criptografía como Herramienta de Seguridad

Es la herramienta de carácter más general que está a disposición de los usuarios y de los diseñadores del sistema.

En una computadora aislada, el sistema operativo puede determinar de manera fiable quiénes son el emisor y el receptor de todas las comunicaciones interprocesos, ya que el sistema operativo controla todos los canales de comunicaciones de la computadora. En una red de computadoras, la situación es bastante distinta. Una computadora conectada a la red envía y recibe bits desde el exterior y no tiene ninguna forma inmediata y fiable de determinar qué maquina o aplicación han recibido/enviado esos bits. Si bien hay direcciones de red que permiten saber quién envía o recibe, puede haber suplantación de identidad por lo que no se puede fiar de las mismas. Por ejemplo, una computadora maliciosa podría enviar un mensaje con una dirección de origen falsificada y otras computadoras podrían recibirlo.

Por tanto, la única alternativa es eliminar, de alguna manera, la necesidad de confiar en la red; este es el trabajo de la criptografía. Desde un punto de vista abstracto la criptografía se utiliza para restringir los emisores y/o receptores potenciales de un mensaje. La criptografía moderna

se basa en una serie de secretos, denominados claves, que se distribuyen selectivamente a las computadoras de una red y se utilizan para procesar mensajes. La criptografía permite al receptor de un mensaje verificar que el mensaje ha sido creado por alguna computadora que posee una cierta clave: esa clave es el origen del mensaje. De forma similar, un emisor puede codificar su mensaje de modo que sólo una computadora que disponga de cierta clave pueda decodificar el mensaje, de manera que esa clave se convierte en el destino. Las claves están diseñadas de modo que no sea computacionalmente factible calcularlas a partir de los mensajes que se han generado con ellas, ni a partir de ninguna información pública. Las claves están diseñadas de modo que no sea computacionalmente factible calcularlas a partir de los mensajes que se hayan generado con ellas, ni a partir de ninguna otra información pública.

Bases de la Criptografía

El cifrado es un medio de restringir los posibles receptores de un mensaje. Un algoritmo de cifrado permite al emisor de un mensaje garantizar que sólo pueda leer el mensaje una computadora que posea cierta clave.

Un algoritmo de cifrado consta de los siguientes componentes:

- Un conjunto K de claves
- Un conjunto M de mensajes
- Un conjunto C de mensajes de texto cifrado
- Una función E: K-> (M->C). Es decir, para cada k ε K, E(k) es una función para generar mensajes de texto cifrado a partir de los mensajes de texto plano. Tanto E como E(k) para cualquier k deben ser funciones computables de manera eficiente.
- Una función D: K -> (C->M). Es decir, para cada k ε K, D(k) es una función para generar mensajes de texto plano a partir de los mensajes de texto cifrado. Tanto D como D(k) para cualquier k deben ser funciones eficientemente computables.

Un algoritmo de cifrado debe proporcionar esta propiedad esencial: dado un mensaje de texto cifrado c ε C, una computadora puede calcular m tal que E(k)(m) = c sólo si posee D(k). Así, una computadora que posea D(k) puede descifrar los mensajes de texto cifrado para obtener los mensajes de texto plano que se usaron para producirlos, pero una computadora que no posea D(k) no puede descifrar esos mensajes cifrados. Puesto que los mensajes cifrados están generalmente expuestos es importante que no se pueda deducir D(k) a partir de los mensajes cifrados.

Existen dos tipos principales de algoritmos de cifrado:

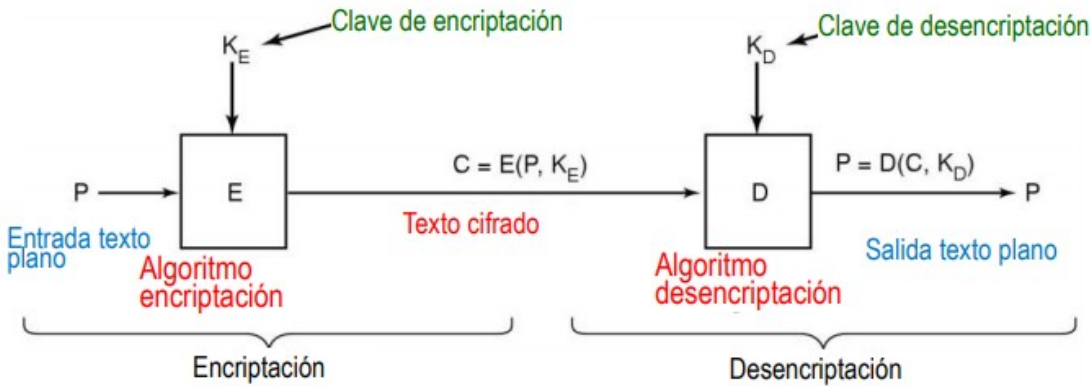
⇒ **Algoritmo de cifrado simétrico**: En un algoritmo de cifrado simétrico, se utiliza la misma clave para cifrar y descifrar, es decir, E(k) puede deducirse a partir de D(k) y viceversa. Por tanto, es necesario proteger el secreto de E(k) con el mismo grado que el de D(k).

⇒ **Algoritmo de cifrado asimétrico**: En un algoritmo de cifrado asimétrico, las claves de cifrado y descifrado son distintas.

Resulta impracticable, desde el punto de vista computacional, deducir $D(k_d, N)$ a partir de $E(k_e, N)$ por lo que no es necesario mantener $E(k_e, N)$ en secreto y dicha clave puede ser ampliamente diseminada; por tanto, $E(k_d,N)$ es la clave publica y $D(k_d,N)$ es la clave privada. N es el producto de dos números primos de gran magnitud p y q aleatoriamente seleccionados. El algoritmo de cifrado es $E(k_e,N)(m)=m^{ke}$ mod N, donde k_e satisface la ecuación $k_e k_d$ mod (p-1)(q-1) = 1. El algoritmo de descifrado será entonces $D(k_d,N)(c)=c^{kd}$ mod N.

El uso del mecanismo de cifrado asimétrico comienza con la publicación de la clave pública del destino. Para la comunicación bidireccional, el origen debe también publicar su clave pública. Esa “publicación” puede ser tan simple como entregar una copia electrónica

de la clave, o puede tratarse de un mecanismo más complejo. La clave privada (o "clave secreta") debe ser guardada celosamente, ya que cualquiera que disponga de esa clave podrá descifrar cualquier mensaje creado a partir de la correspondiente clave pública.



Relación entre el texto plano y el texto cifrado

Criptografía con clave secreta

- Secreta Sustitución Monoalfabética: cada letra es reemplazada por otra letra diferente
- Clave de encriptación dada, fácil de obtener la clave de desencriptación
- Clave criptográfica secreta llamada clave criptográfica simétrica

Criptografía con clave Pública

- Todos los usuarios toman un par de claves: una clave pública y una clave privada publica la clave pública no publica la privada
- La clave pública es la clave de encriptación (depende……)
- La clave privada es la clave de desencriptación

Firma Digital

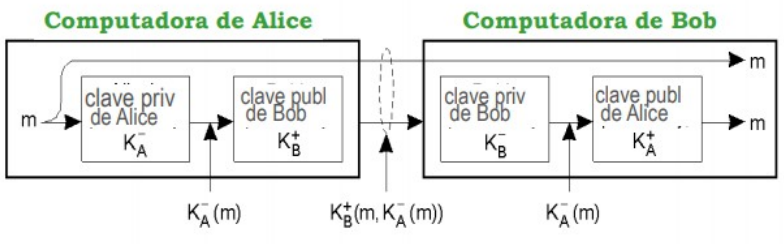
El proceso de restringir el conjunto de potenciales emisores de un mensaje se denomina autenticación. La autenticación es, por tanto, complementaria al cifrado. La autenticación sirve para demostrar que un mensaje no ha sido modificado.

Un algoritmo de autenticación consta de los siguientes componentes

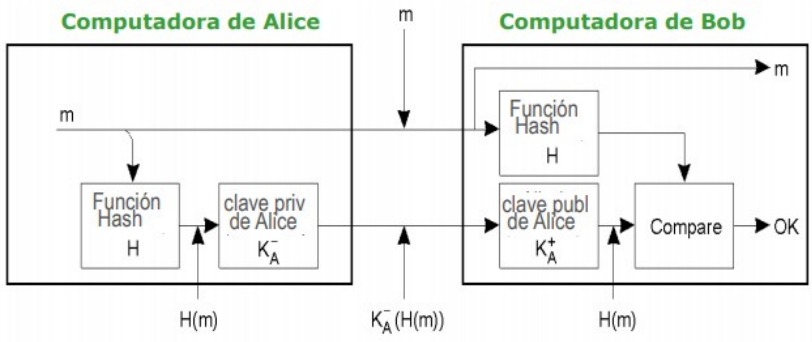
- Un conjunto K de claves.
- Un conjunto M de mensajes.
- Un conjunto A de autenticadores.
- Una función S: K -> (M->A). Es decir, para cada k ε K, S(k) es una función para generar autenticadores a partir de los mensajes. Tanto S como S(k) para cualquier k deben ser funciones computacionalmente eficientes.
- Una función V:K -> (MxA -> {true,false}). Es decir, para cada k ε K, V(k) es una función para verificar autenticadores de mensajes. Tanto V como V(k) para cualquier k deben ser funciones completamente computables.

La propiedad critica que un algoritmo de autenticación debe poseer es esta: para un mensaje m, una computadora puede generar un autenticador a ε A tal que V(k)(m,a)=true sólo si posee S(k). Así, una computadora que posea S(k) podrá generar autenticadores para los mensajes de modo que cualquier otra computadora que posea V(k) pueda verificarlo. Sin embargo, una computadora que no tenga S(k) no podrá generar autenticadores para los mensajes que puedan

verificarse utilizando V(k). Puesto que los autenticadores están generalmente expuestos no debe ser factible deducir S(k) a partir de los autenticadores.
Al igual que hay dos algoritmos de cifrado, hay también dos variedades principales de algoritmos de autenticación. El primer tipo de algoritmo de autenticación utiliza cifrado simétrico. En un código de autenticación de mensajes (MAC, message-authentication code) se genera una suma de comprobación criptográfica a partir del mensaje utilizando una clave secreta. El conocimiento de V(k) y el conocimiento de S(k) son equivalentes: podemos derivar una función a partir de la otra, por lo que es necesario mantener en secreto el valor de k. Debido a la propiedad de resistencia a las colisiones de la función hash, podemos estar razonablemente seguros de que ningún otro mensaje permita crear el mismo valor MAC. Un algoritmo de verificación apropiado será entonces V(k)(m,a) = (f(k,m)=a)
El segundo tipo principal de algoritmo de autenticación es el algoritmo de firma digital, y los autenticadores generados por uno de estos algoritmos se denominan firmas digitales. En un algoritmo de firma digital, no resulta computacionalmente factible deducir $S(k_s)$ a partir de $V(k_v)$; en particular, V es una función de una sola dirección. Por tanto, k_v es la clave pública y K_s es la clave privada.



Firma digital – Criptografía con clave publica



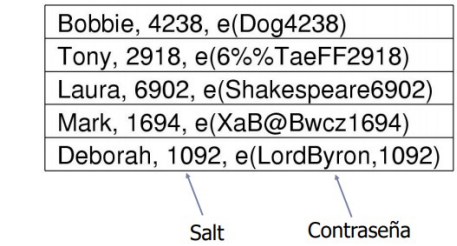
Firma digital – utilización de digesto

Autenticación de Usuario

Generalmente, la autenticación de usuario se basa en uno o más de tres cuestiones: la posesión de algo por parte del usuario (una clave o tarjeta), el conocimiento de algo (un identificador de usuario y una contraseña) por parte del usuario y/o un atributo del usuario (huella digital, patrón retinal o firma).
La autenticación de usuario es crucial para identificar correctamente al usuario, dado que el sistema de protección depende del user ID.

1. **Contraseñas**: El método más habitual para autenticar la identidad de un usuario consiste en usar contraseñas. Cuando el usuario se identifica a sí mismo mediante un ID de usuario o un nombre cuenta, se le pide una contraseña. Si la contraseña suministrada por el usuario coincide con la contraseña almacenada en el sistema, el sistema supone que el propietario de la cuenta está accediendo a la misma. Las contraseñas pueden considerarse un caso especial de las claves o de las capacidades.

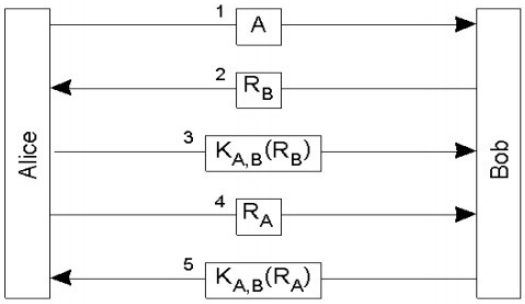
2. **Vulnerabilidad de contraseñas**: Las contraseñas son extremadamente comunes porque son fáciles de comprender y utilizar. Lamentablemente, a menudo pueden adivinarse, ser mostradas por accidente, ser interceptadas o ser ilegalmente transferidas por un usuario autorizado a otro que no lo está. Para permitir que las contraseñas permanezcan en secreto, se las debe cambiar frecuentemente y no se deben usar contraseñas adivinables.
3. **Computadoras cifradas**: Un problema que se plantea es la dificultad de mantener en secreto la contraseña dentro de la computadora. Los sistemas UNIX utilizan el cifrado para evitar la necesidad de mantener en secreto su lista de contraseñas. El sistema contiene una función que es extremadamente difícil de invertir, pero fácil de calcular. El problema de este método es que el sistema ya no tiene el control sobre las contraseñas. Aunque las contraseñas se cifren, cualquiera que disponga de una copia del archivo de contraseñas puede ejecutar rutinas de cifrado rápido, cifrando cada palabra en un diccionario, por ejemplo, y comparando los resultados con las contraseñas almacenadas.
4. **Contraseñas de un solo uso**: Por otro lado, para evitar los problemas de intercepción de contraseñas un sistema podría usar un conjunto de contraseñas emparejadas. Cuando se inicia una sesión, el sistema selecciona aleatoriamente una pareja de contraseñas y presenta una parte de la misma; el usuario debe suministrar la otra parte. En este sistema de contraseña de un solo uso, la contraseña es diferente en cada caso. Cualquiera que capture la contraseña de una sesión e intente reutilizarla en otra sesión no tendrá éxito. Las contraseñas de un solo uso constituyen un método para impedir las autenticaciones erróneas debidas a la exposición de la contraseña.



El uso del salt para derrotar la precomputación de las contraseñas encriptadas.

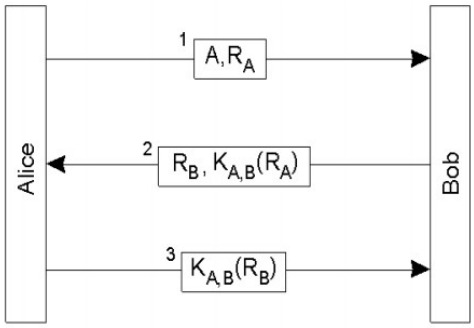
Autenticación usando contraseñas

Cinco mensajes para el protocolo



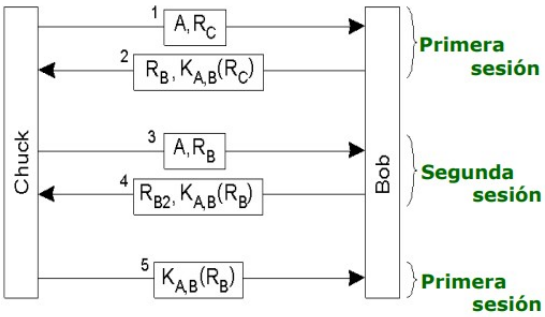
Autenticación emisores basada en claves compartidas

Tres mensajes para el protocolo



Autenticación emisores basada en clave secreta compartida

Tres mensajes para el protocolos. Problema: ataque por reflejo.



Autenticación emisores basada en clave secreta compartida

Contrafuegos para proteger los sistemas y redes

Para conectar una computadora de confianza a una red que no sea de confianza, podemos utilizar un contrafuegos. Un contrafuegos es una computadora, dispositivo o enrutador que se sitúa entre el sistema seguro y el que no lo es. Un contrafuegos de red limita el acceso a la red entre los dos dominios de seguridad y monitoriza y registra todas las conexiones. También puede limitar las conexiones basándose en la dirección de origen o de destino, el puerto de origen o de destino o la dirección de la conexión.

Un contrafuegos permite dividir la red en múltiples dominios. Una implementación habitual define varios dominios distintos: el dominio no seguro constituido por Internet, otro dominio semi-seguro, denominado zona desmilitarizada (DMZ, demilitarized zone); y un tercer dominio formado por las computadoras de la empresa. Las conexiones entre Internet y las computadoras DMZ, y entre las computadoras de la empresa e Internet se permite, pero no se permiten las comunicaciones entre Internet o las computadoras DMZ y las computadoras de la empresa.

Cabe aclarar que los contrafuegos no impiden los ataques de tipo túnel, es decir, los ataques contenidos dentro de los protocolos o conexiones que el contrafuegos permita. Por ejemplo, un contrafuegos no detendrá un ataque por desbordamiento de un búfer a un servidor web, ya que la conexión http está permitida y es el propio contenido de la conexión http lo que se utiliza como mecanismo de ataque. Del mismo modo, los ataques por DOS pueden afectar a los contrafuegos al igual que a otras máquinas. Otra vulnerabilidad de los contrafuegos es la suplantación.

Un contrafuegos persona es una capa de software incluida en el sistema operativo o que se añade como una aplicación. En lugar de limitar la comunicación entre dominios de seguridad, limita la comunicación a un determinado host. Por ejemplo, un usuario puede añadir un

contrafuegos personal a su PC con el fin de denegar a un caballo de Troya el acceso a la red a la que esté conectado el PC. Un contrafuegos del tipo proxy de aplicación entiende los protocolos que utilizan las aplicaciones a lo largo de la red, como por ejemplo SMTP, que usa para la transferencia de correo electrónico. Los contrafuegos de llamadas al sistema se ubican entre las aplicaciones y el kernel, monitorizando la ejecución de las llamadas al sistema.

Clasificaciones de Seguridad

El departamento de defensa de los EE.UU especifica cuatro clasificaciones de seguridad para los sistemas: A, B, C y D.

La clasificación de nivel más bajo es la división D, o protección mínima. La división D incluye solo una clase y se usa para los sistemas que no cumplan los requisitos mínimos de ninguna de las otras clases de seguridad. Por ejemplo, MS-DOS y Windows 3.1 pertenecen a la división D.

La división C, el siguiente nivel de seguridad, proporciona una protección discrecional y mecanismos de atribución de responsabilidad a los usuarios y control de sus acciones, mediante el uso de capacidades de auditoria. La división C tiene dos niveles C1 y C2. Un entorno C1 es aquél en el que una serie de usuarios cooperantes acceden a un conjunto de datos con el mismo nivel de confidencialidad. La mayoría de las versiones de UNIX son de clase C1. Un sistema C2 añade un control de acceso de nivel individual a los requisitos de un sistema C1. Por ejemplo, los derechos de acceso de un archivo pueden especificarse en el nivel de un solo individuo. Además, el administrador del sistema puede auditar selectivamente las acciones de un solo individuo. Algunas versiones seguras especiales de UNIX han sido certificadas con el nivel C2.

Los sistemas de protección obligatoria de la división B tienen todas las propiedades de un sistema de clase C2 y además asignan un nivel de seguridad de cada objeto. Está divida en B1, B2 y B3.

La clasificación de nivel superior es la división A. Estos sistemas usan técnicas formales de diseño y verificación para asegurar la seguridad.

Seguridad Multinivel

La seguridad multinivel, también conocida por las siglas MLS (del inglés Multiple Level of Security), es un tipo de política de seguridad que clasifica a los usuarios en distintos niveles de seguridad, permitiendo el acceso simultáneo a distintos usuarios con diferentes permisos y asegurándose de que cada usuario acceda a aquellos recursos que tiene autorizados y de la forma que tiene autorizada. Por tanto es una tecnología que permite proteger secretos y así evitar que algún usuario pueda acceder a cierta información cuyo acceso no tiene permitido.

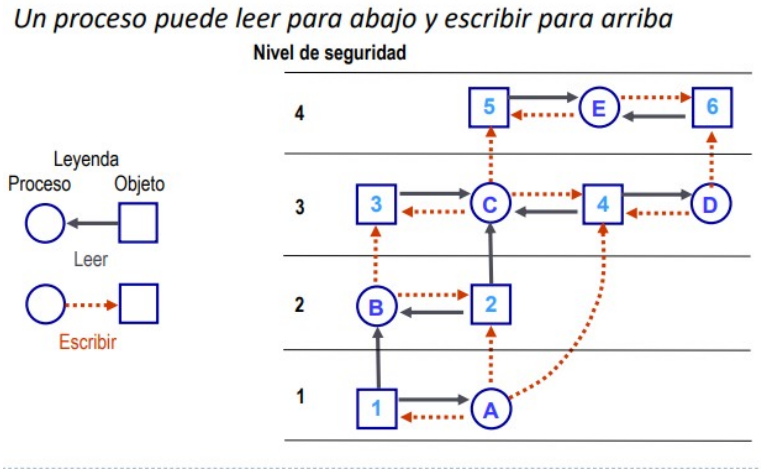
Podemos extender el concepto de seguridad multinivel desde un computador individual a redes de computadores. A la aplicación de la seguridad multinivel en ese entorno se le llama sistemas inter-dominio o CDS (del inglés Cross Domain Systems). En ella hablamos de dominios en lugar de niveles y los datos se comparten en las redes de comunicaciones.

El modelo de seguridad Bell-Lapadula, consiste en dividir el permiso de acceso de los usuarios a la información en función de etiquetas de seguridad. Se basa en seguridad multinivel.

Este modelo se centra en la confidencialidad y no en la integridad por lo que también se lo conoce como modelo de confidencialidad. Se distinguen 2 tipos de entidades, sujetos y objetos. Se define estados seguros y se prueba que cualquier transición se hace de un estado seguro a otro. Un estado se define como estado seguro si el único modo de acceso permitido de un sujeto a un objeto está en concordancia con la política de seguridad. Para determinar si un modo de acceso específico está permitido, se compara la acreditación de un sujeto con la clasificación del objeto para determinar si el sujeto está autorizado para el modo de acceso especificado. El esquema de clasificación/acreditación se expresa en términos de un retículo. El modelo define 2

reglas de control de acceso mandatorio (MAC) y una regla de control de acceso discrecional (DAC) con 3 propiedades:

- Propiedad de seguridad simple: Un sujeto de un determinado nivel de seguridad no puede leer un objeto perteneciente a un nivel de seguridad más alto.
- Propiedad (Propiedad estrella): Un sujeto de un determinado nivel de seguridad no puede escribir un objeto perteneciente a un nivel de seguridad más bajo. (También llamada propiedad de confinamiento).
- Propiedad de seguridad discrecional: Se utiliza una matriz de acceso para especificar el control de acceso discrecional.

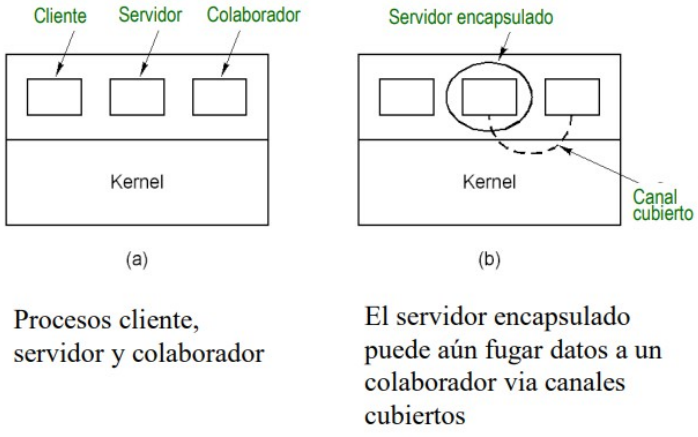


El método Biba se basa en seguridad multinivel y permite garantizar la integridad de los datos. Se basa en dos principios que son opuestos al método de Bell-La Padula.

- Write-Down. El proceso puede escribir solamente objetos en su nivel de seguridad o inferior.
- Read-Up. El proceso puede leer solamente objetos de su nivel de seguridad o más alto.

Canales Encubiertos

Un canal encubierto, es un canal que puede ser usado para transferir información desde un usuario de un sistema a otro, usando medios no destinados para este propósito por los desarrolladores del sistema. Para que la comunicación sea posible suele ser necesario un preacuerdo entre el emisor y el receptor codifique el mensaje de una forma que el receptor sea capaz de interpretar.



Canal encubierto

Canales Cubiertos

Hay un tipo especial de canales encubiertos a los que se les llama canales cubiertos (en inglés side channel) en los que el emisor de forma accidental comunica al receptor. En estos sistemas el desafío del receptor es descubrir y decodificar el canal oculto para obtener la información.

Esteganografía

Uno de los métodos más fáciles de implementar es el de inyección o agregado. Consiste esencialmente en agregar o adosar al final de un archivo, de cualquier tipo, otro archivo que será el contenedor del "mensaje a ocultar", también de cualquier tipo. Esta metodología es la más versátil, pues permite usar cualquier tipo de archivo como portador
(documentos, imágenes, audio, vídeos, ejecutables, etc) y adosar el "paquete enviado", que es otro archivo, también de cualquier tipo.
Esta es una técnica que no se vale de las limitaciones humanas (vista y oído) para implementar la estrategia esteganográfica, sino que se vale de la forma de funcionamiento de la aplicaciones software que utilizan el portador. No degradan el contenido del portador de ninguna forma, por ejemplo, si es una imagen, permanecerá intacta; ya que el "mensaje" se le inyecta o adosa al final de la misma y la aplicación usada para visualizarla la mostrará normalmente hasta donde ella finalice. Esto es debido que todo tipo de archivo, en su cabecera, entre otros, contiene ciertos bytes fijos (en cantidad y ubicación) usados exclusivamente para indicar el tamaño del archivo. La aplicación que utilice un archivo, de cualquier tipo, siempre lee su cabecera primero, adquiere ese valor como su tamaño (en cantidad de bytes) y seguidamente lee el resto del archivo hasta el final indicado por dicho valor. De modo que si se coloca algo (mensaje) más allá del valor de ese parámetro, no será leído por la aplicación normal, por tanto no detectado, y el archivo portador funcionará normalmente.
Si bien es la técnica más sencilla de implementar, y de uso muy difundido, tiene la gran desventaja que provoca crecimiento del portador, tanto como el tamaño de su mensaje, siendo por tanto una estrategia fácilmente detectable. Otra desventaja, aunque muy relativa y eventual, es que el crecimiento del portador podría ser limitante a la hora de trasferirlo por las redes, particularmente por Internet.
Si no se requiere reunir requisitos de indetectabilidad es uno de los métodos preferidos, por su sencillez, flexibilidad y escasas limitaciones. Prácticamente cualquier tipo de portador es admitido, con o sin compresión, incluso módulos ejecutables. En algunos casos provoca corrupción del portador, lo cual no es gran problema: practicada la técnica e inyectado el mensaje se prueba el portador con su aplicación correspondiente, si se ha degradado y/o no funciona bien, sencillamente toma otro, del mismo u otro tipo y se repite la operación.