MODULO 5

ADMINISTRACIÓN DE MEMORIA CENTRAL

CONTENIDO:
Distintas técnicas de administración de la memoria central

OBJETIVO DEL MÓDULO: Describir las distintas técnicas empleadas en la Administración y gestión del Almacenamiento Principal (Memoria Central).

OBJETIVOS DEL APRENDIZAJE: Después de estudiar este módulo, el alumno deberá estar en condiciones de:

- Explicar el Funcionamiento y reconocer los distintos administradores de la memoria central.
- Conocer las funciones de la administración y gestión de la memoria central
- Conocer las ventajas y desventajas de cada técnica de Administración de la memoria central
- Distinguir los distintos tipos de algoritmos utilizados en memoria virtual.
- Conocer y explicar la terminología específica empleada en éste módulo.

Metas:

La meta de este módulo es describir las distintas técnicas empleadas en la Administración y Gestión del Almacenamiento Primario (Memoria Central), como así también entender el funcionamiento y poder reconocer los distintos Administradores y sus funciones, las ventajas y desventajas de cada Administrador y distinguir los distintos tipos de algoritmos utilizados en Memoria Virtual y además, familiarizarse con la terminología específica empleada.

5. ADMINISTRACIÓN DE MEMORIA CENTRAL (MC).

Recomendamos la lectura del Anexo 5.A1.3 al final de este módulo a los efectos de repasar previamente algunos temas necesarios para una correcta comprensión de éste módulo.

En un sistema **monoprogramado**, la **memoria central** se divide en dos partes: una para el S.O. (monitor residente, núcleo) y otra para el programa que se ejecuta en ese instante.

En un sistema **multiprogramado**, la parte de "usuario" de la memoria debe subdividirse más para que tengan lugar varios procesos. La tarea de subdivisión la lleva a cabo dinámicamente el S.O. y se conoce como **Gestión de Memoria**.

En un sistema **multiprogramado** resulta vital una gestión efectiva de la memoria. Si sólo hay unos pocos procesos en memoria, entonces la mayor parte del tiempo estarán esperando a la E/S y el procesador estará desocupado. Por esto, se debe repartir eficientemente la memoria para que contenga tantos procesos como sea posible.

En la memoria se almacenan los programas y los datos. Para poder ejecutar un programa necesariamente debe residir en la memoria central o física del computador. De allí el procesador toma las instrucciones, las procesa y devuelve los resultados nuevamente a la memoria. En algunas arquitecturas (con DMA) se intercambian directamente los datos con el módulo de E/S. Las características del **tiempo de acceso** y **el tiempo de ciclo de memoria** tienen una gran influencia sobre la velocidad del procesamiento. Se define como tiempo de acceso al que transcurre entre el inicio y el fin de una operación de lectura o escritura sobre una locación de memoria. El tiempo de ciclo de memoria es el retraso que impone el hardware entre el fin de una operación y el comienzo de la siguiente.

La memoria es el centro de las operaciones de un sistema de computación. La memoria es un arreglo de palabras o bytes, cada uno con su dirección propia. El procesador toma instrucciones de la memoria de acuerdo al contenido del Registro contador de programa de la CPU que oficia de puntero a cada una de las locaciones o posiciones que posee la memoria. Estas instrucciones pueden causar cargas y almacenamientos adicionales a direcciones de memorias específicas.

Los módulos de administración de memoria en un S.O. se refieren a la administración de la memoria primaria, principal o central que tiene un equipo computacional. *Memoria primaria o Central*[1] es aquella accedida directamente por los procesadores para obtener sus instrucciones y datos que requiere el procesamiento de un programa.

Está compuesto por un hardware de direccionamiento, las memorias cachés, y la memoria RAM y ROM. Adoptamos el término de memoria central para designar las cachés y las RAM y ROM como una unidad.

La organización del almacenamiento constituye un factor importante para el funcionamiento y el rendimiento general de la computadora. Las estrategias pueden ser varias, pero cada una se propone cumplir con los objetivos lo mas eficientemente posible contemplando las restricciones de costos, overhead, complejidad de los módulos del S.O., hardware adicional para direccionamiento, tablas, protección, etc..

Se busca un máximo de eficiencia en el uso del sistema. Para ello se presentan distintos esquemas que van desde sistemas monoprogramados a multiprogramados y desde memoria real hasta memoria virtual. A continuación, analizaremos los distintos esquemas de administración de la memoria central. La selección de un determinado esquema para un sistema específico depende de muchos factores, esencialmente del diseño del hardware.

5.1. Funciones del Administrador de la Memoria Central

Las principales funciones son:

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

[1] Usaremos el término Memoria Principal o Memoria Central como sinónimos

Desde el punto de vista de la administración, la memoria central, es un vector finito de bytes que debe ser repartido entre varios procesos.

La única forma de ejecutar un proceso es que el bloque de instrucciones que contiene y los datos necesarios se encuentren en memoria central.

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

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

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

5.2. Objetivos de la Administración de la Memoria Central

Como servicios a los programas en ejecución, los principales objetivos que debe cubrir un Administrador de memoria central son:

a) la Protección, b) el uso compartido de códigos y datos, c) la reubicabilidad de la información, que comprende las generaciones de direcciones y el mapeo de la memoria, y d) proveer direcciones lógicas y físicas.

A continuación detallaremos estos puntos.

a) Protección

Entre sí cuando varios procesos comparten la Memoria. Si la memoria es un recurso compartido se debe proteger los accesos contra:

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

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

1. **datos puros** que se protege contra Lectura (R) o contra escritura (W) o ambos (R/W).
2. **código:** contra lectura (poco frecuente) y contra ejecución (X).
3. **pila (Stack):** R (pop), W (push) y X.

Además se distingue que el proceso:

a) no tenga permiso de acceso (se verifica la tabla de permisos y si el proceso no tiene permiso de acceso, es terminado).
b) no pueda efectuar la operación en ese instante (condiciones de concurrencia, en este caso el proceso se bloquea).

Normalmente, un programa usuario no puede acceder a ninguna porción del sistema operativo, ya sea programa o información. De la misma forma, usualmente un programa en un proceso no puede saltar a una instrucción en otro proceso. Sin arreglos especiales, un programa o un proceso no pueden acceder al área de datos de otro proceso. El procesador debe ser capaz de abortar cualquiera de estas instrucciones en el punto de ejecución.

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

b) Uso compartidos de Códigos y datos (una sola copia en memoria central):

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

Definimos como ***códigos reentrantes o puros*** a aquel código de programa que no se modifica, es decir, no tiene ni usa variables globales, sino sólo locales y la almacena en una pila (stack). En otras palabras, no usa variables estáticas.

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

c) Reubicabilidad:

En un sistema multitarea, la memoria central está compartida por muchos procesos. Generalmente, no es posible para el programador saber qué otros programas están residentes en

memoria central en el tiempo de ejecución de su programa. Además, se busca poder trasladar procesos activos dentro y fuera de la memoria central para maximizar la utilización del procesador, proveyéndolo de varios procesos listos para ejecutar. Una vez que un programa fue sacado de la memoria central al disco, sería difícil establecer cuál es el próximo en volver a la memoria central, y debería ser colocado en la misma región de la memoria central que antes.

Brevemente, se puede decir, que la memoria es un recurso utilizado para el almacenamiento de las instrucciones que forman un proceso. El ciclo típico de ejecución de instrucciones incluye la transferencia de instrucciones de memoria a CPU, decodificación de instrucciones, búsqueda de operandos y ejecución efectiva de las instrucciones, posteriormente los resultados también pueden ser almacenados en memoria.

Cuando se ejecuta un proceso, este debe ser cargado en memoria previamente. Anteriormente, cuando es un programa binario ejecutable esta almacenado en memoria secundaria, típicamente en disco. Cuando un proceso entra en ejecución se utiliza un proceso del sistema que se denomina cargador, que lo transfiere a memoria. En Unix por ejemplo, el cargador almacena el proceso en memoria distinguiendo en el tres segmentos distintos:

- Segmento de código
- Segmento de Datos
- Pila

Dichos segmentos se distribuyen según muestra la siguiente figura.

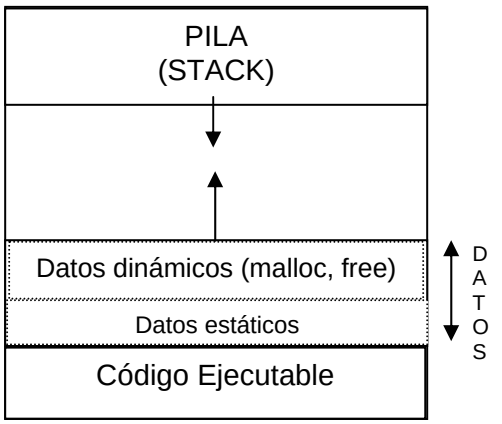


Figura 5.0 Organización de la M.C.

La asignación de memoria en los segmentos es responsabilidad de diferentas etapas en el sistema.

- Compilación
- Enlazado (linker)
- Carga (loader)
- Asignación de memoria en tiempo de ejecución (run time)

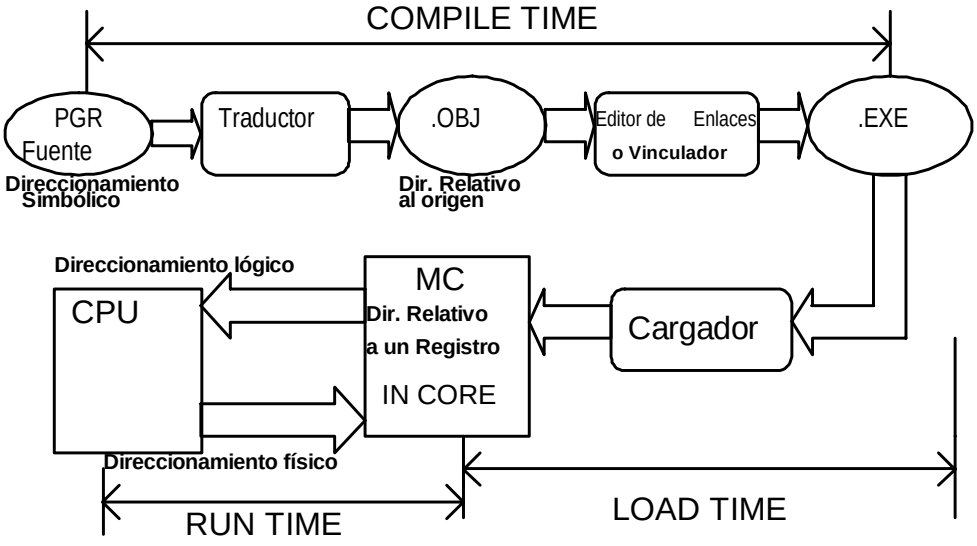


Fig. 5.1 Traducciones de direcciones de un programa

El Sistema Operativo debe conocer la ubicación de la información del control del proceso y de la pila de ejecución, además de la primera ubicación en donde comenzar el proceso. Como el S.O. es responsable de mover el proceso a la memoria central, las direcciones son fáciles de mover. Sin embargo, el procesador debe negociar con referencias de memoria del programa. Las instrucciones de ramificación (**branch instructions**) apuntan a la próxima instrucción a ejecutar, las instrucciones de referencia de datos (**reference to data**) apuntan al dato referenciado por el programa. Sin embargo, el procesador y el S.O. deben ser capaces de trasladar las referencias de memoria encontradas en el programa en la actual ubicación de la memoria física.

A lo largo de la vida de un programa las direcciones de memoria son representadas de distintas formas. Como programa fuente, desde que se lo crea, las **direcciones** son **simbólicas** como por ejemplo la variable x. El "Compilador" lo transforma en **direcciones reubicables** como por ejemplo 354 bytes desplazados del origen, o sea del módulo principal del programa. El "Loader" al cargarlo a memoria central lo transforma en **direcciones absolutas** como por ejemplo ABF23 en hexadecimal, el cual es un paso necesario para su ejecución. Estas direcciones se conocen como **direcciones lógicas** que luego se convierten durante la ejecución a **direcciones físicas o absolutas** mediante un ***mapper*** que oficia de traductor**.**

A continuación se desarrollarán las distintas instancias que sufren las direcciones.

Enlace de direcciones: Cada locación de memoria lleva implícita dos cosas: su **dirección y** su **contenido**. Como para ejecutar un programa es necesario que esté cargado en memoria, se debe establecer cual será su **dirección de inicio** y **cuanto ocupará**. De esto se deducen dos cosas: Salvo que se utilice alguna técnica especial, el tamaño del programa estará limitado al tamaño de la memoria, y es necesario establecer cual es el espacio de memoria direccionable al que el usuario puede acceder normalmente para trabajar.

Respecto de esto último, podemos decir que la primera dirección de la memoria puede ser la cero, sin embargo no necesariamente esta sea la primer dirección accesible para el usuario.

En un programa, el origen lo establece el compilador, además que las direcciones suelen ser simbólicas y también suele ser el compilador el encargado de realizar el enlace entre estas direcciones simbólicas y las direcciones de localización en la memoria, mientras que el cargador se ocupa de volverlas direcciones absolutas de memoria.

Cabe desatacar que el enlace de direcciones puede hacerse tanto en tiempo de compilación, como de carga, como de ejecución del programa. En cada caso habrá distintas ventajas y desventajas a tener en cuenta.

1) Generación de direcciones (Address binding)

- ***Compile time:*** Generadas por el traductor (Compilador y vinculador). No se sabe en que dirección física va a residir el programa por lo tanto se generan direcciones relativas al origen.

Si en el momento de la compilación se sabe donde residirá el programa en memoria puede generarse código absoluto. Si mas tarde cambia la posición de inicio, será necesario volver a compilar el código. Esto es muy incomodo.

Ejemplo: Los programas de MS-DOS con formato ".COM", se enlazan de manera absoluta durante la compilación.

- ***Load time:*** El traductor genera código relativo al comienzo del proceso y deja una constante sin resolver para la Tabla de símbolos que será generada por el **vinculador** o **editor de enlaces** de la forma "constante de reubicación más desplazamiento" en cada entrada de la Tabla de símbolos. Luego se generan las direcciones absolutas con el cargador que conoce la constante de reubicación y el hardware hace las operaciones de suma.

➢ **CARGA DINÁMICA:**

Se la utiliza para obtener una mejor utilización del espacio de memoria. Una rutina no se carga hasta que se le llame.

Todas las rutinas se almacenan en disco en formato de carga relocalizable. El programa principal se carga en memoria y se ejecuta. Cuando una rutina llama a otra, la rutina que llama primero comprueba si se ha cargado la otra. Si no es así, se llama al cargador de enlace relocalizable para que cargue en memoria la rutina deseada y actualice las tablas para reflejar este cambio. Entonces el control se transfiere a la rutina recién cargada.

Ventajas:

a) Nunca se carga una rutina que no se usa.
b) Útil cuando se requieren grandes cantidades de código para manejar situaciones que ocurren con poca frecuencia. Ejemplo: Rutinas de error.
c) Aunque el tamaño total del programa puede ser grande, la porción que realmente se utiliza puede ser menor, es decir, la que se carga.

Desventajas:

a) La construcción de programas para aprovechar este esquema.

➢ **ENLACE DINÁMICO:**

El concepto de enlace dinámico es semejante al de carga dinámica, en vez de postergar la carga hasta la ejecución, se posterga el enlace.

Esta característica se emplea generalmente con las bibliotecas del sistema.

En el contenido binario se incluye un *stub (*fragmento) para cada referencia a una rutina de la biblioteca del sistema, el cual es un pequeño trozo de código que indica como localizar la rutina de

biblioteca residente en memoria. Al ejecutar este fragmento se reemplaza así mismo con la dirección de la rutina y la ejecuta. La siguiente vez que se llega a ese segmento de código la rutina se ejecuta directamente, sin costos adicionales por el enlace dinámico.

Ventajas:

a) Se utiliza generalmente con las bibliotecas del sistema. Sin este recurso todos los programas de un sistema necesitan incluir en su contenido ejecutable una copia de la biblioteca, desperdiciando espacio en disco y en memoria central.
b) Todos los procesos que utilizan una biblioteca de lenguaje ejecutan solo una copia del código de la biblioteca.
c) Esta característica puede extenderse para incluir actualizaciones a bibliotecas (corrección de errores). Una nueva versión puede sustituir a una biblioteca y todos los programas que hacen referencia a la biblioteca usaran automáticamente la nueva versión. Sin el enlace dinámico, todos estos programas se tendrían que enlazar nuevamente para tener acceso a la nueva biblioteca (bibliotecas compartidas).

2) OVERLAYS (Superposiciones):

Objetivo: Que un proceso pueda ser mayor que la cantidad de memoria que se le asigna.
La idea es conservar en memoria solo aquellas instrucciones y datos que se requieren en un momento determinado. Cuando se necesitan otras instrucciones, se cargan en el espacio que antes ocupaban las que ya no se requieren (se sobre escriben).

Ventajas:

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

Desventaja:

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

Es una técnica que busca mantener en la memoria aquellas instrucciones o datos que se necesitan en un momento determinado. Su objetivo es reducir la problemática de la limitación que encontramos en el tamaño de los programas respecto del tamaño de la memoria. Si más tarde se necesitaran nuevas instrucciones o datos que no están cargados en la memoria, estos pueden ser cargados en los espacios que aquellos que de dejen de ser útiles dejen libres.

- ***Run time:*** El código puede ser relocalizado durante su ejecución. Generalmente requiere un hardware especial.

En un sistema multiprocesos, la memoria central disponible es compartida entre procesos. Por esto el programador no puede saber de ante mano que otros procesos estarán residentes en memoria cuando su programa se encuentre en estado de ejecución. Para poder maximizar el uso del procesador, los procesos son enviados o sacados de memoria (“swap in” y “swap out”, en ingles). Cuando un proceso es sacado, seria una limitación que cuando este sea enviado nuevamente a memoria, ocupe la misma región en esta.

Como no podemos saber dónde estará ubicado el proceso, debemos permitir que este se mueva en la memoria para permitir el swapping. El sistema operativo debe conocer la ubicación del bloque de control del proceso y el stack, así como también el punto de entrada al mismo. El procesador se encargara de las referencias a memoria, instrucciones de salto y referencias a datos. De alguna manera el sistema operativo y el hardware del procesador se encargaran de traducir las referencias a memoria del programa en direcciones físicas, reflejando la ubicación actual del proceso en memoria.

3). Mapas de Memoria Central (mapper):

En la Figura 5.02 respecto a la Reubicabilidad y traducciones de direcciones, se presenta un mapa de memoria central real en que el programa, cuya longitud es N (su espacio de direcciones), debe caber en el espacio de memoria libre M. El mecanismo de traducción de direcciones verifica esta condición y si el programa es mayor que el espacio libre lo rechaza. Esta situación se considera una restricción que se resuelve con la memoria virtual (Fig. 5.03).

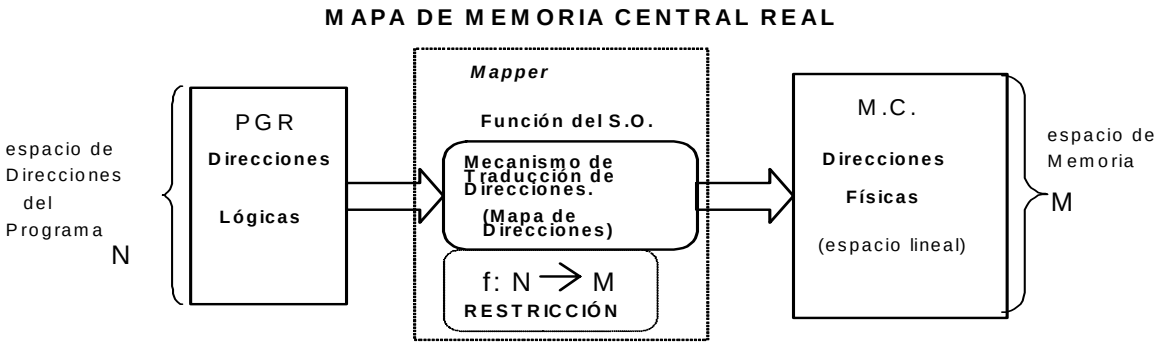


Figura 5.02 Reubicabilidad y traducciones de direcciones: mapa de memoria real

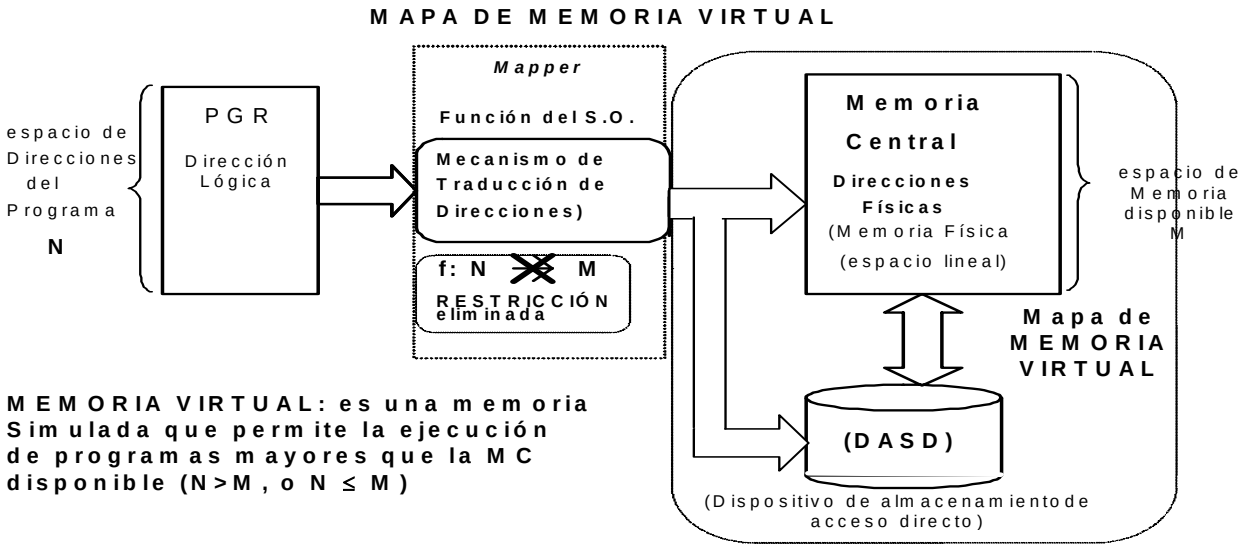


Figura 5.03 Reubicabilidad y traducciones de direcciones: mapa de memoria virtual

d) Direcciones físicas y direcciones lógicas

La mayoría de las técnicas de administración de memoria central distinguen dos tipos de direcciones:

- **direcciones lógicas:** Emitidos por el proceso cuando direcciona su espacio de memoria. Su valor más bien, no refleja la organización de la memoria física de la computadora.
- **direcciones físicas**: Provienen de adaptar la dirección lógica emitida por el programa en ejecución al espacio de direccionamiento real de la computadora.

La memoria central físicamente está organizada en un espacio de direccionamiento lineal, o unidireccional, compuesto por una secuencia de bytes o palabras. La memoria secundaria, a su nivel físico, esta organizada de una manera similar.

Los programas están organizados en módulos o funciones, algunos no modificables, de solo lectura y ejecución, y otros que contienen datos que podrían modificarse. La organización modular de los programas trae una serie de ventajas:

- Los módulos pueden ser escritos y compilados independientemente.
- Con un pequeño overhead adicional, se puede proveer distintos niveles de protección a módulos distintos.
- Es posible la introducción de mecanismos para poder compartir los módulos entre procesos.

Definimos:

- **Espacio de direccionamiento lógico:** Conjunto de todas las direcciones lógicas generadas por un programa en ejecución.
- **Espacio de direccionamiento físico:** Conjunto de todas las direcciones físicas correspondientes a las direcciones lógicas generadas en el instante de ejecución.

En cuanto a la organización física, la memoria de una computadora esta organizada en al menos dos niveles: memoria central y secundaria. La memoria central provee un acceso rápido a un relativo alto costo. Este tipo de memorias son de lectura y escritura (RAM, Caché, etc.). La memoria secundaria es más lenta y barata, y usualmente discos o memorias Caché mas lentas. Las memorias secundarias de

gran capacidad son usadas para un almacenamiento permanente de programas y datos, mientras que la memoria central guarda los programas y datos que actualmente están en uso.

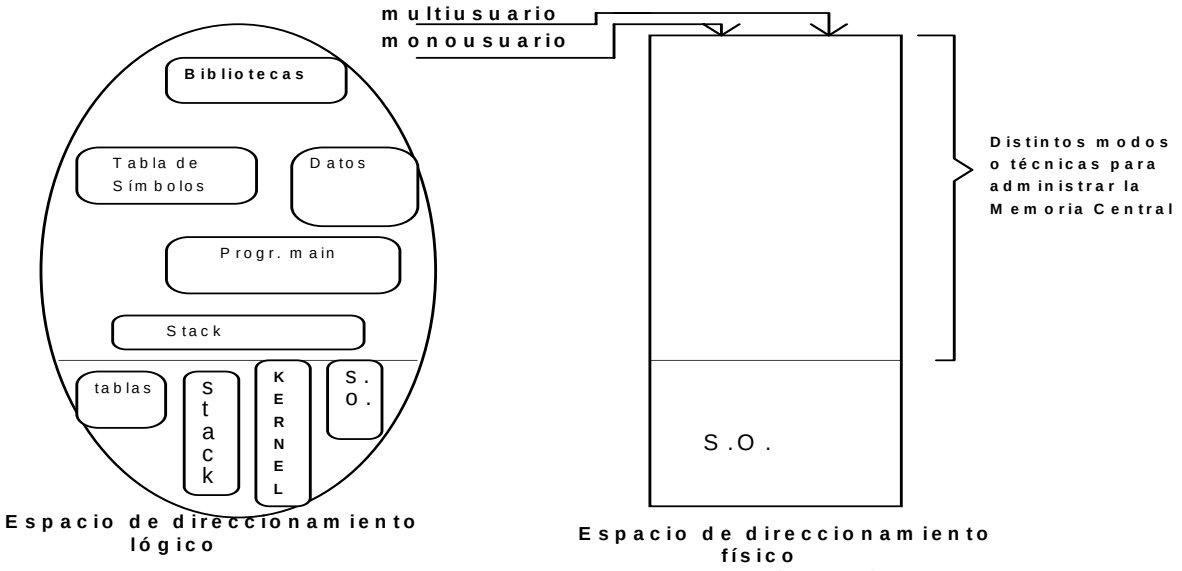


Figura 5.04 Espacio de direccionamiento físico y lógico

Cuando usamos el esquema de enlace de direcciones en tiempo de ejecución, las direcciones lógicas y físicas no coinciden. Para convertir las direcciones lógicas de un programa origen a direcciones físicas de memoria, se necesita una unidad de administración de memoria que se encargue de llevar a cabo la relación.

Por ejemplo, supongamos que los programas del usuario deban comenzar a partir de la dirección física 100 de la memoria. Consideramos a esta dirección como el registro base a partir del cual el usuario tiene acceso a la memoria. Para el usuario, esto es transparente, por lo que él siempre creerá que las direcciones de memoria comienzan a partir de las dirección física 000. La unidad de administración de memoria convertirá las direcciones lógicas del programa de usuario, sumándolas al registro base. Si la CPU indica que debemos acceder a la dirección lógica 044, que por estar por debajo de los 100 sabemos que no corresponde a una dirección accesible por el usuario, se calculará la dirección física 100+044=144, a través de la suma del registro base y la dirección lógica que tenemos.

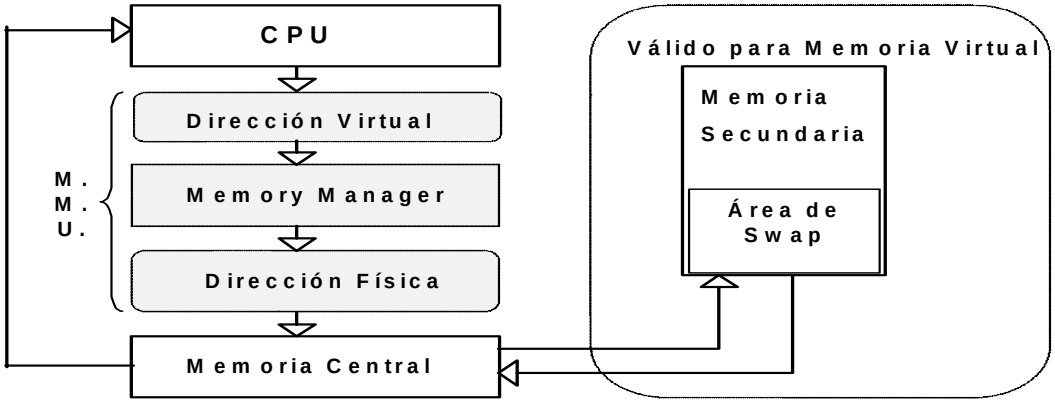


Figura 5.05 Modulos que intervienen en la ejecución de un programa y lo sombreado corresponde al M.M.U.

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

La correspondencia en el tiempo de ejecución de direcciones físicas y lógicas, se logra por la **Unidad de Administración de Memoria (M.M.U. Memory Manager Unit)**, el cual es un dispositivo de hardware incorporado en el procesador, que posterga los enlaces hasta la ejecución. Un programa de usuario nunca ve las direcciones físicas reales, trata con direcciones lógicas, el hardware de correspondencia de memoria convierte las direcciones lógicas en físicas. La localidad final de la dirección no se determina hasta que no se efectúe la referencia.

En la figura 5.05 se esquematiza los módulos que intervienen para lograr la ejecución de un programa. Recordamos que en memoria central se involucran las memorias caché (internas y externas), RAM y ROM.

Las políticas elegidas para la administración de la memoria pueden estar influidas por el deseo de mantener los módulos de administración de memoria lo mas pequeños y simples posibles, o por la necesidad de aumentar la flexibilidad del usuario o la eficiencia del sistema.

Las distintas técnicas que describimos en este módulo son:

a) Técnicas de administración sin swapping:

- Administración sin S.O.
- Administración de una sola memoria contigua.
- Administración de memoria particionada.
- Administración de memoria particionada de reubicación.
- Administración de memoria paginada.
- Administración de memoria segmentada.

b) Técnicas de administración con swapping:

- Memoria virtual con paginación por demanda y
- Memoria Virtual Mixta: segmentación con paginación por demanda.

5.3. TÉCNICAS DE ADMINISTRACIÓN SIN SWAPPING

Las políticas elegidas para la administración de la memoria pueden estar influidas por el deseo de mantener los módulos de administración de memoria lo más pequeños y simples posibles, o por la necesidad de aumentar la flexibilidad del usuario o la eficiencia del sistema.

Estas Técnicas se basan en que el programa debe ser cargado en memoria central sin que se produzcan intercambios de porciones del programa residentes en el disco con la memoria central y viceversa.

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

5.3.1. MEMORIA DEDICADA (MÁQUINA DESNUDA SIN S.O.)

Las primeras computadoras no poseían S.O. por lo que los programadores debían directamente acceder al hardware y gestionar la memoria en sus programas.

Actualmente se pueden diseñar algunos sistemas específicos, como ser una máquina automática, o un instrumento de medición, comandado por un sistema embebido en que el software gobierna su funcionamiento. Las características de este esquema son:

- No hay S.O.
- La traducción del código binario debe ser realizado manualmente. Se cargan los datos y el programa mediante switches o manualmente mediante un teclado hexadecimal y se arranca la ejecución poniendo en el Program Counter la primer dirección del programa a ejecutar.
- El manejo de los recursos lo hace el programa ya que es posible reemplazar los servicios brindados por el S.O.: mediante la programación en lenguaje de máquina por el usuario. Luego no se necesita el soporte del S.O.
- El usuario tiene toda la memoria central. disponible. Todas las acciones de control y toda la responsabilidad.

Ventajas:

- Máxima flexibilidad, Máxima simplicidad y mínimo costo.

Desventajas:

- No existen servicios para interrupciones y errores.
- Baja productividad para el usuario y el Hardware, su programación es engorrosa y compleja. Es imposible la construcción de programas largos o grandes y la multiprogramación.

Ejemplo de empleo de este tipo de administración son los S.O. de robots, ascensores, instrumentos, equipos industriales, etc..

5.3.2. ASIGNACIÓN CONTIGUA SIMPLE O MONITOR RESIDENTE.

Fue el primer esquema en la evolución de lo S.O.. La asignación contigua simple es un sencillo plan de administración de memoria que no requiere características especiales de Hardware y se asocia usualmente con pequeños computadores con sencillos sistemas operativos del tipo por lotes (Batch) y monousuarios como el MS-DOS.

La memoria está dividida conceptualmente en tres campos contiguos como se observa en la figura 5.06. Una porción está asignada permanentemente al S.O. El resto de la memoria está disponible

y se asigna al único trabajo que está procesando. Aunque el trabajo solo use físicamente una porción de esta memoria, toda la memoria le es asignada. Las características son:

- Es monoprogramado.
- Se divide la memoria en 3 áreas.
- Comúnmente, se pone el S.O. en la zona baja de memoria, cerca del vector de interrupciones.
- A veces, se usa un mecanismo primitivo de protección para evitar que se modifique la zona del S.O. provista por un registro como se observa en la Fig. 5.06-

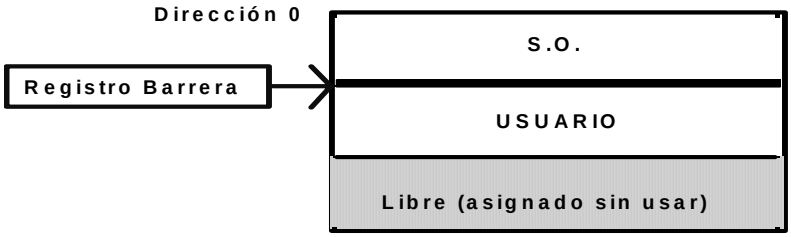


Figura 5.06 Esquema de administración de memoria de monitor residente

El mecanismo de protección de Hardware es de tipo primitivo para asegurar que los programas del usuario no interfieran en forma accidental o deliberada con el S.O.. Entonces se debe proteger el código y los datos del S.O. frente a cambios provocados por los procesos de los usuarios. Además se necesita proteger a los procesos de los usuarios entre ellos.

Con cada dirección generada por la CPU, el Hardware (el M.M.U. y no el S.O.), sigue el siguiente algoritmo:

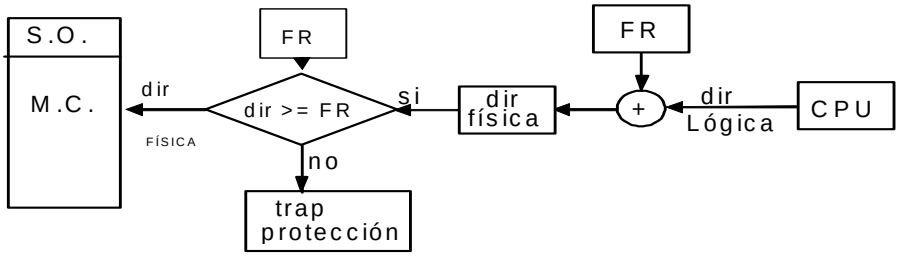


Figura 5.07 Hardware de protección

El mecanismo consiste en usar un registro barrera, o base, o de defensa (***fence register*** y un modo dual de trabajar la CPU: - supervisor / usuario -). El registro base contiene la dirección del área protegida (incluyendo al S.O.). Por cada referencia a memoria, el hardware hace comprobaciones para asegurarse que no se trate de un acceso al área protegida. De intentarse tal acceso, ocurre una Excepción ***(trap***) y el control se transfiere al S.O. que aborta la ejecución.

Se usa vinculación en tiempo de carga (load time binding). Las direcciones son relativas al registro base (fence register FR).

El valor de este registro debe ser estático (invariante) durante todo el tiempo de ejecución del proceso, ya que sus direcciones fueron establecidas en el tiempo de carga, de acuerdo al valor cargado en el Registro Base (***fence register***).

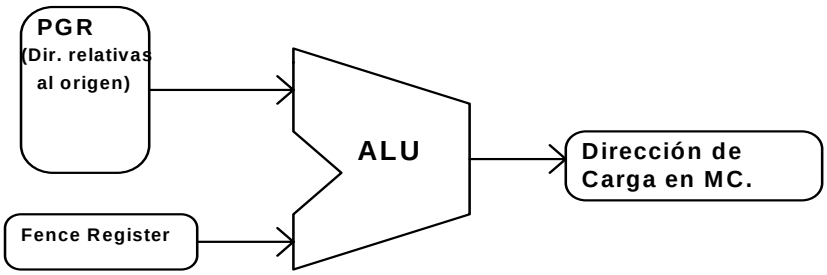


Fig. 5.08 Proceso de carga de direcciones de un programa

Al registro base se suma a cada dirección generada por un proceso de usuario en el momento de enviar la memoria como se indica en la figura 5.08.

A veces, es necesario cambiar el tamaño del S.O. (sacar códigos de drivers no usados, agrandar el tamaño de estructuras internas, etc.). Este esquema provee un camino que permite cambiar el tamaño (y por consiguiente la posición base), durante la ejecución de un programa. Por ejemplo: el S.O. contiene código y espacio de almacenamiento temporal para manejadores de dispositivos, si un manejador de dispositivo no se usa habitualmente entonces no es deseable conservar en memoria sus datos y código,

a este código se lo llama **transiente** (temporal) del SO, el cual viene y va según se requiera. Esto puede provocar que el programa usuario se sobrescrita.

Soluciones:

1. Cargar el programa usuario en memoria alta (high-memory).
2. Resolver direcciones en tiempo de ejecución (runtime): podemos mover el programa dentro de la memoria central y cambiar el FR (Fence Register).

La carga de los programas de usuario es otro problema a considerar, aunque el espacio de direccionamiento del computador empieza en 00000, la primera dirección del programa usuario es la primera dirección tras el fence register, pudiendo afectar a las direcciones que utiliza el programa usuario. Si la dirección acotada por este registro se conoce al compilar, puede generarse un código con direcciones absolutas, éste empezará en el valor conocido como límite entre el S.O. y la 1er localidad libre disponible y se extenderá a partir de allí. Luego si cambia la dirección acotada, será necesario recompilar este código. Como alternativa, el compilador puede generar código **reubicable**, donde la asignación se retrasa hasta el momento de la carga. Si la dirección acotada cambia, el código de usuario sólo precisa ser cargado de nuevo para incorporar este nuevo valor. En ambos casos el valor acotado será **estático** durante la ejecución del programa. El valor acotado por el fence register puede moverse solamente cuando no hay ningún programa usuario ejecutándose.

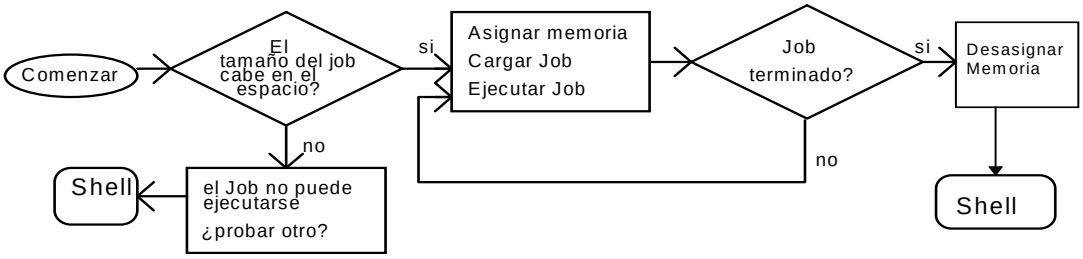


Fig. 5.09 Algoritmo del administrador de memoria con monitor residente

Ventajas:

- Simplicidad. Un S.O que use este plan puede requerir apenas un KBy, en su totalidad comparados con los 256 KBy o más que necesitan los más complejos.
- Facilidad de comprensión y uso.
- Es muy simple de implementar.
- El S.O es sencillo, pequeño y produce poco overhead de direccionamiento.
- El Hardware de protección y direccionamiento es poco costoso y complejo.

Desventajas:

La principal desventaja estriba en que la memoria no se utiliza al máximo. Se advierten tres ineficiencias:

- Una parte de la memoria no se usa en absoluto y está asignada al programa del usuario. Por Ejemplo la memoria libre es de 6 MeBy y el programa ocupa 500 kiBy se desperdician 5,5 MeBy
- En la mayoría de los sistemas con canales o DMA, hay ocasiones en que la CPU no está usando activamente alguna porción de la memoria.
- Algunas porciones de memoria del espacio de dirección del usuario contienen información que jamás se accesa o se usa. Por ejemplo las rutinas de error.

Otra desventaja es la falta de flexibilidad, por ejemplo, el espacio de dirección del trabajo debe ser más pequeño que la memoria central pues de lo contrario no se pueden correr los trabajos. Además, no permite la multiprogramación.

Resumen de las desventajas: Mal uso de la memoria. Hay zonas que no se emplean en absoluto. No soporta multiprogramación. Espacio de direccionamiento del usuario acotado por el tamaño de la memoria disponible.

5.3.3. ASIGNACIÓN PARTICIONADA

La asignación particionada, en sus distintas formas, es una de las más sencillas técnicas de administración de memoria que permite la multiprogramación. La multiprogramación permite el acceso del procesador a varios procesos simultáneamente ejecutando de a uno por vez según las técnicas que se estudiaron en los módulos precedentes. Para poder repartir el procesador entre varios procesos, necesitamos disponer de ellos en la memoria central. Para ello la memoria central se divide en distintas **regiones o particiones de memoria***,* cada una de las cuales mantiene un espacio de dirección para un solo trabajo o proceso (figura 5.10).

Se necesita muy poco Hardware especial para apoyar éste tipo de asignación; es suficiente un mecanismo que impida que un trabajo o proceso afecte al S.O. o a otros trabajos residentes en la memoria, sea en forma accidental o intencional. Se podrían usar dos registros límites para encerrar la

partición que se está usando; en caso de que el trabajo tratara de acceder a la memoria fuera de la partición, ocurriría una interrupción de protección. Sin embargo, los registros límites deben cambiarse cada vez que se reasigna el procesador a otro proceso.

Se necesitan básicamente dos registros límite, uno inferior o barrera, también llamado Registro Base (**RB**) y otro superior (**LS**) o Registro Límite (**RL**), para evitar que los procesos accedan a direcciones fuera de su espacio de direccionamiento como se observa en la Fig. 5.10 para el proceso P2.

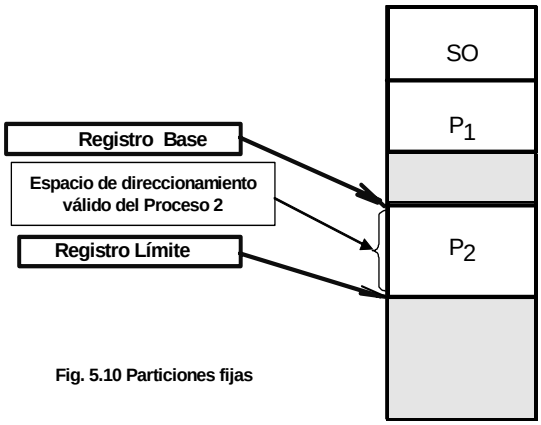


Fig. 5.10 Particiones fijas

Esto se esquematiza en el siguiente diagrama:

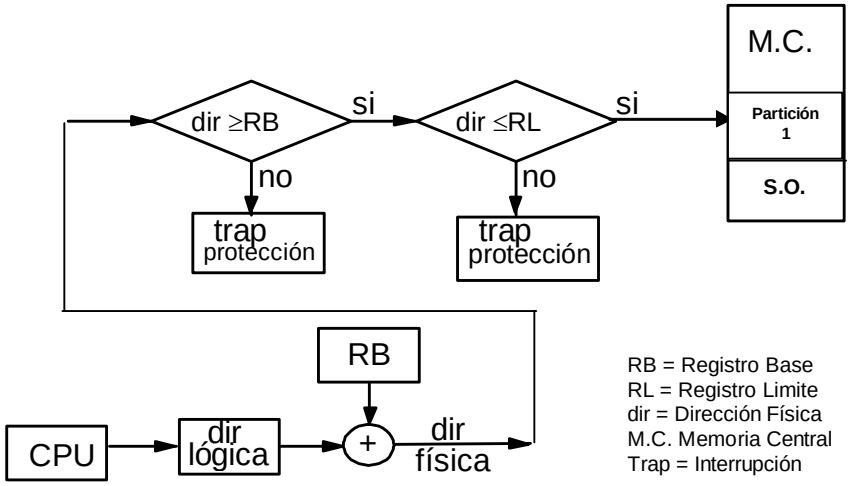


Fig. 5.11 Protección de accesos a memoria por registros base y límite

Primero relocalizamos dinámicamente la dirección lógica sumándole el valor del registro base para obtener la dirección física. El registro base contiene el valor de la menor dirección física en que se ubica el proceso; el registro limite contiene el mayor valor del intervalo de direcciones físicas. Con el registro base y limite cada una de las direcciones lógicas generadas por la CPU debe ser menor o igual al registro limite y mayor o igual al registro base y luego de esta verificación esta dirección relocalizada se envía a memoria.
Hay dos variantes de esta técnica.

1. Particiones fijas

La especificación de la partición la puede designar el usuario o se puede incorporar en el S.O. Cada paso de trabajo proporcionado por un usuario debe especificar la máxima cantidad de memoria necesaria. Entonces se encuentra y asigna una partición de tamaño suficiente.

El algoritmo del S.O. que maneja la asignación y desasignación es simple. Se divide la memoria en particiones de tamaño fijo. Cada partición contiene un proceso. De la cola de trabajos se escoge un proceso que quepa en alguna de las particiones libres. Este esquema fue usado originalmente por OS/360 MFT (Multiprogramming with a Fixed Number of Tasks) con una cola de Jobs por cada partición, que luego se reemplazó por una sola cola.

El nivel de multiprogramación esta limitado por el número de particiones. Cuando una partición está libre se selecciona un proceso de la cola de entrada y se carga en la partición libre, cuando el proceso termina, la partición está disponible para otro.

La técnica de especificación estática es especialmente apropiada cuando se conocen bien los tamaños y frecuencia de los trabajos. En tales casos se escogen los tamaños de las particiones para que correspondan lo mejor posible a los tamaños de trabajos más comunes. Sin embargo, se puede desperdiciar mucha memoria si no se conocen los tamaños y frecuencias de los trabajos o si son diversos. Un trabajo que precise **m** palabras de memoria puede ejecutarse en una partición de **n** palabras, donde **n ≥ m**. La diferencia entre estos números (n – m) es el **fragmento** de la memoria de una partición que no se utiliza.

2. Particiones variables

Particiones son creadas durante la ejecución de los procesos para ajustarse a sus necesidades. Las particiones se crean durante el procesamiento del trabajo para que se hagan corresponder los tamaños de las particiones con los de los trabajos. Los planes de asignación de partición dinámica se emplean en muchos casos que no incluyen al S.O. Las tablas deben constituirse con entradas para cada área libre y cada partición asignada, especificando el tamaño, posición y restricciones de acceso a cada partición (ver Figura 5.12).

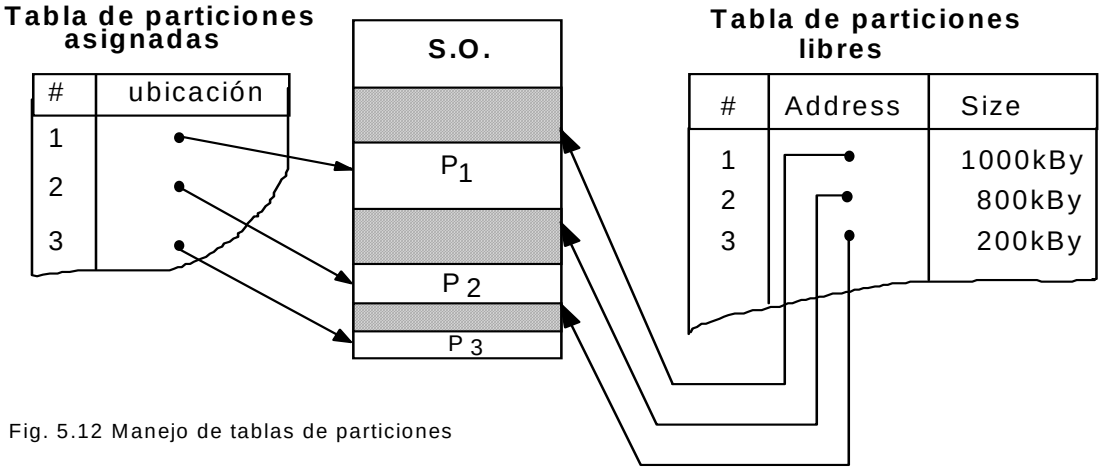


Fig. 5.12 Manejo de tablas de particiones

- **Tamaño determinado por la longitud del proceso a ejecutar:** Usada por OS360 y 370 MVT

Inicialmente se usaron dos tablas distintas (ver Figura 5.12), una para las áreas asignadas y otra para mantener el estado de las áreas no asignadas o libres.

Las tablas representan una forma de llevar el control del uso de la memoria. El número de entradas necesarias en la tabla de particiones libres y ocupadas no es constante, depende del número de áreas libres y ocupadas que existan en cualquier momento. Por lo tanto, el enfoque de llevar dos tablas simples es algo ineficiente y en las implementaciones reales se simplificó utilizando una sola tabla con una columna de estado.

Esta tabla tiene las siguientes características:

- ✓ Una sola en el S.O.
- ✓ Contiene todas las áreas no usadas y usadas.
- ✓ El tamaño queda determinado por un límite inferior (Registro Base) y un límite superior (Registro Límite)
- ✓ Se usa para buscar espacio libre a los nuevos procesos recorriendo la columna y viendo los tamaños.
- ✓ Cuando los procesos entran al sistema se colocan en una cola de entrada, el S.O toma los requisitos de memoria de cada proceso y la cantidad de espacio de memoria en los cuales el proceso será localizado. Le crea una entrada a la tabla con su identificación, registro base, registro límite, tamaño, y marca esa porción de memoria como ocupada.
- ✓ Con el tiempo se agregó una columna para protección (Lectura o Escritura) y una columna de modificación para grabar las modificaciones ocurridas durante la ejecución en memoria secundaria.
- ✓ Cuando un proceso termina, libera memoria, El S.O. la marca como libre y luego busca otro proceso de la cola de entrada (si hay alguno esperando y el requerimiento de espacio cabe) y le asigna ese espacio marcándolo como ocupado la porción requerida de memoria y crea una nueva entrada en la tabla como espacio libre del tamaño del fragmento no utilizado.
- ✓ La memoria se asigna a los procesos hasta que se pueda satisfacer los requerimientos de espacio que formulan los mismos.

- ✓ El S.O. espera la liberación de un bloque de tamaño suficiente para revisar la cola de entrada y ver si algún proceso tiene requisitos de memoria que se ajusten a ese bloque.
- ✓ El número de entradas en la tabla de partición de áreas libres y ocupadas no es constante, depende del número de particiones y áreas libres que existan en un momento determinado.

Inicialmente se carga el S.O. residente y queda toda la memoria restante disponible para los procesos del usuario y se lo considera como un gran bloque de memoria disponible o un ***hole*** (hueco). Cuando llega un proceso y necesita memoria, se construye un bloque de tamaño suficiente para ese proceso, creando dos entradas en la tabla, una del tamaño asignado que se anota en la columna como espacio ocupado y la otra como espacio disponible. Cuando la memoria está completamente ocupada por los procesos, queda solo una pequeña porción de memoria sin ocupar. A partir de este instante, si algún proceso se completa y abandona la partición que ocupaba, el S.O. marca en la columna el estado de la partición como libre y va a buscar un proceso de la cola de ejecución. Si lo encuentra, solo se asigna la cantidad de memoria requerida, quedando disponible el resto del fragmento para solicitudes futuras.

Cuando un proceso termina, libera memoria, el S.O. debe analizar en la tabla si hay un fragmento aledaño libre tanto inferior o superior, entonces modifica la entrada sumando esas porciones libres como si fuera una sola partición.

El S.O. espera hasta que un bloque lo suficientemente grande se libere y puede saltar a la cola de entrada para ver si hay requisitos de memoria más chico de algún otro proceso.

- ***Fragmentación externa***

Al seleccionar un algoritmo de memoria particionada se deben tener en cuenta varios factores, entre ellos el más importante es, sin duda, el efecto de la fragmentación: el desarrollo de un número grande de pequeñas áreas libre por separado (es decir, que la memoria libre total se fragmenta para constituir pedazos pequeños entre áreas asignadas).

Se puede minimizar el problema de la fragmentación en la asignación particionada mediante una cuidadosa selección de los algoritmos específicos a usar. O bien, superar la fragmentación usando técnicas distintas, como lo son la reubicación y paginación.

En general para la fragmentación externa se pueden considerar los siguientes puntos:
Es la generación de un gran número de áreas libres, separadas debido al constante intercambio de trabajos, entonces se producen huecos muy pequeños en los cuales no cabe ningún proceso. Estas porciones no son aprovechadas por lo que se desperdician y a éste desperdicio se lo llama ***fragmentación externa***.

Existe fragmentación externa cuando hay memoria suficiente para un requerimiento pero no es contigua, sino que está distribuida en pequeñas áreas libres de la memoria central que no es aprovechable. Dependiendo de la capacidad total de la memoria y del tamaño promedio de los procesos, la fragmentación externa puede ser un problema mayor o menor. Ejemplo: El análisis estadístico del primer ajuste revela que dados *n* bloques asignados se perderán otros *0,5 n* bloques debido a la fragmentación, es decir, la mitad de la memoria puede estar utilizada. A esto se lo conoce como **Regla del 50%.**

Otro problema que surge con este esquema es si se considera un hueco de 10.000 bytes y el proceso solicita 9998 bytes, si lo asignamos queda un hueco de 2 bytes. El procesamiento adicional para administrar el hueco será bastante mayor que el propio hueco.

Para resolver el problema de fragmentación externa se recurre a ***la compactación*** que es un proceso del S.O. que reorganiza las zonas de memoria ocupadas de forma tal de dejar en una sola porción o área libre (o sea, poner contigua las zonas libres y "borrar" los límites).

Sólo se puede compactar si la ubicación es dinámica y se efectúa en el momento de la ejecución, ya que para que los procesos puedan ejecutarse en sus nuevas posiciones, hay que relocalizar todas las direcciones internas. Si la relocalización es estática y se efectúa durante el ensamblado o la carga, la compactación no puede realizarse. Si las direcciones se relocalizan dinámicamente, solo se requiere mover los programas y los datos y luego cambiar el registro base para reflejar la nueva dirección base.

Cuando compactar: ¿En qué instante se toma la decisión de compactar? Inmediatamente después que se desocupa una partición. Se mantiene siempre una única área libre.

a) Pros: Asignación es trivial, reduce las tablas que hay que llevar.
b) Contras: Repetidas y a veces innecesarias compactaciones.

Instancias para compactar:

1. Sólo compactar cuando es necesario o sea cuando aparece un pedido de asignación que no pueda ser satisfecho.
2. Cuando la compactación es posible, debemos determinar su costo. El algoritmo de compactación más sencillo consiste en mover todos los procesos hacia un extremo de la memoria, todos los huecos se mueven en la dirección contraria produciendo un gran hueco de memoria disponible. Este

esquema puede ser bastante costoso. Es bastante difícil seleccionar una estrategia de compactación óptima.

Adelantando un poco un concepto importante: el swapping. Es importante considerar que se puede combinar el intercambio con la compactación. Un proceso puede descargarse de la memoria central por intercambio al almacenamiento auxiliar para reincorporarse más tarde. Cuando el proceso se descarga, se libera su memoria y quizás se reutilice para algún otro proceso. Cuando el proceso tiene que reincorporarse por intercambio, pueden aparecer varios problemas. Por ejemplo:

a) Si se utiliza la relocalización estática, el proceso debe reincorporarse exactamente a las mismas posiciones de memoria que antes ocupaba. Esta restricción puede requerir que todos los demás procesos se descarguen para liberar esa memoria.
b) Si la relocalizacion es dinámica, entonces el proceso se puede reincorporar a una localidad diferente. Encontramos un bloque libre, compactando si es necesario, e reincorporamos el proceso.

Una estrategia para la compactación consiste en descargar los procesos que hay que mover, e reincorporarlos a localidades de memoria diferentes. Si los intercambios o la técnica de descargar e reincorporar los procesos ya forma parte del sistema operativo el código adicional para la compactación pueden ser mínimos.

Proceso de Asignación y desasignación.

1. Se busca un área lo suficientemente grande para el proceso a cargar en la Tabla de particiones Libres (ver Figura 5.12). El área seleccionada se divide en dos, una resulta asignada y la otra queda libre.
2. La asignación, en esta técnica, entra dentro de los esquemas de asignación dinámica de almacenamiento que consiste en resolver el problema de satisfacer un requerimiento de tamaño **n** dado un conjunto de áreas donde esa asignación es posible. Para ello se consulta a las tablas.
3. Estrategias de asignación:
   - **Primer ajuste** o ***First-fit*** (**FF**): Recorre la tabla de áreas libres y asigna el primer área donde quepa el requerimiento. La búsqueda puede comenzar en el inicio del conjunto de áreas o a partir de donde terminó la búsqueda del primer hueco anterior.
   - **Mejor ajuste** o ***Best-fit*** (**BF**): Utiliza Estrategia Global. Busca el área más chica que pueda contener el requerimiento. Lo mejor es tener la tabla de espacios libres ordenada por tamaño (de otra manera se debe recorrer toda la tabla). Si existe una región exacta del tamaño buscado le es asignada al programa, lo cual no es necesariamente cierto en first fit. Evita la separación de áreas grandes y la fragmentación. Produce áreas libres muy pequeñas.
   - **Peor ajuste** o ***Worst-fit*** (**WF**): Asigna el bloque más grande posible. Produce áreas libres más grandes que los otros dos. Es el esquema complementario del BF. Se debe buscar en toda la tabla, a menos que esté ordenada por tamaño. Al producir el área sobrante más grande ésta puede ser más útil que un área más pequeña.
   - **Ajuste rápido** o ***Next Fit*** **(NF):** Es la más utilizada. Estrategia local. Es igual al BF con un puntero de localización. Se tiene una tabla con ***n*** entradas, donde la primera sería un puntero a la cabeza de una lista de huecos de un tamaño dado, la segunda a otra lista de otro tamaño y así sucesivamente.

     Empíricamente, se demostró que FF y BF son algoritmos mejores que WF en términos de tiempo y aprovechamiento de la memoria y, en general, FF es más rápida que BF y NF es más rápido que BF y aprovecha mejor la Memoria Central que FF.
4. Recuperación de Memoria Central: Procura unir la partición devuelta con algún área adyacente para obtener una región libre mayor.

Ventajas:

1. Mayor grado de multiprogramación y los algoritmos son simples y fáciles de implementar.
2. Hardware poco costoso.

Desventajas:

1. El Hardware de direccionamiento y reubicación hace más lento al sistema.
2. Hay un desperdicio de memoria central por fragmentación externa.
3. Se producen tiempos de compactación prolongados.
4. Se necesita conocer la longitud del programa a priori para la asignación.
5. Más memoria utilizada por el Sistema Operativo (Las Tablas ocupan espacio de memoria).
6. Limita el tamaño del programa a ejecutar al máximo tamaño de memoria central.

Dentro de los esquemas que se utilizan para saber la cantidad de memoria ocupada y libre se encuentran los siguientes:

La Administración de asignación de memoria en particiones variables.

Administración del espacio con mapa de bits:

Este esquema consiste en que la memoria se divide en bloques o frames de cierto tamaño y se mantiene en memoria una tabla en donde cada bit se utiliza para identificar un bloque ocupado y un bloque libre. Si un bloque esta ocupado el bit asociado a él esta en 1, si el bloque esta libre el bit esta en 0.

Un aspecto importante de mencionar en este enfoque es que si el tamaño de asignación es pequeño entonces el mapa de bits es mas grande, comparativamente con un bloque de asignación más grande el mapa de bits es más pequeño. El tamaño del mapa de bits solamente está determinado por el tamaño de la memoria y el tamaño del bloque de asignación.

Administración de memoria con listas enlazadas:

Este enfoque consiste asignar la memoria mediante un bloque de asignación. El control de la memoria ocupada o libre, el sistema la mantiene mediante una lista enlazada en donde cada nodo de la lista esta compuesto por los siguientes campos:

- ✓ Bit que representa un proceso o un hueco
- ✓ Número de bloque de inicio
- ✓ Número de bloques utilizados
- ✓ Puntero al siguiente nodo en la lista

En el caso de tener una lista ordenada por direcciones, la ventaja de este enfoque es que la actualización de la lista es directa, pues cada nodo tiene dos vecinos, cada unos de los cuales es un proceso o un hueco, luego al desocuparse un segmento representado por un nodo en la lista pasa a ser un hueco, el cual podría eventualmente agruparse a un hueco vecino.

La desventaja de este enfoque es que la búsqueda de un hueco libre suficiente de satisfacer a un proceso es lenta. Para solucionar esta lentitud se puede tener dos listas ligadas, una para los huecos y otra para los procesos, sin embargo, esto obliga a mantener estas dos listas, es decir deben actualizarse cuando un proceso termina y cuando se otorga memoria a un proceso.

5.3.4. Paginación Pura o Simple

El problema planteado de necesitar un espacio de direccionamiento contiguo y del procesamiento para lograrlo (compactación) con su exagerado overhead, motivó buscar una técnica que aprovechara mejor la memoria. La paginación es una técnica de gestión que permite asignar memoria en forma no contigua.

Para la administración de memoria paginada se divide el espacio de direccionamiento de cada programa que se pretende ejecutar en porciones iguales llamados *páginas,* e igualmente se divide la memoria física en porciones del mismo tamaño llamados ***bloques, marcos o frames.***

Entonces, proporcionando una posibilidad adecuada de mapeo del hardware se puede colocar una página en cualquier bloque libre existente. Las páginas permanecen lógicamente contiguas (es decir, para el programa usuario) aunque los bloques correspondientes en la memoria física no necesariamente lo sean. Como en la administración por particiones reubicables, los programas de los usuarios no se ven afectados por el mapeo expresado anteriormente ya que éste no tiene un efecto visible sobre el espacio de direccionamiento. Todo esto constituye una manera de evitar el requerimiento de espacio contiguo para el programa que se pretende ejecutar.

Para que el hardware realice el mapeo del espacio de direcciones lógicas a la memoria física, debe haber un registro para cada página por separado que establezca la correspondencia lógica - física. Generalmente se llama ***tablas de página*** **o tablas de mapas de página (MPT** o también **PT)** a esos registros. Dichos mapas pueden ser registros especiales de hardware o una porción reservada de la memoria central que se destina para ese fin.

Ya que es posible reubicar cada página por separado, no se necesita que el área que ocupa un trabajo está completamente contigua en la memoria; el único caso de contigüedad deben ser las posiciones en una sola página. La selección del tamaño de la página tiene un efecto fundamental en el rendimiento y costo en este tipo de administración.

Entonces:

- Se divide la memoria física en regiones del mismo tamaño llamadas frames, marcos o bloques.
- La imagen de memoria del proceso se divide en porciones, todas del mismo tamaño, llamadas **páginas** (aunque la última no se complete).
- El concepto de la separación en páginas es lógico mientras que la de Bloque es físico.

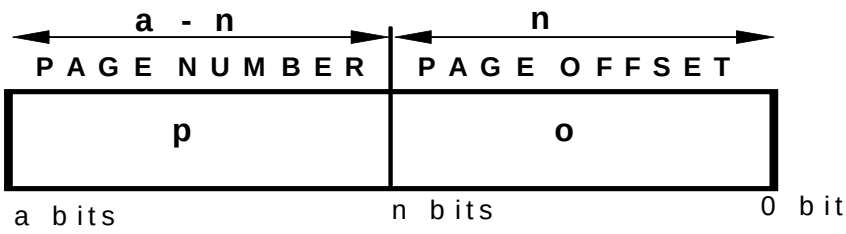


Fig. 5.13 Dirección lógica en paginación

El tamaño de la página y el frame son definida por el Administrador del Sistema y permanece fijo. Típicamente, se toma una potencia de 2 (512 a 4096 Bytes) o sea 2^9 a 2^{12} bits. Si el tamaño de la página es 2^n entonces los n primeros bits (los de menor peso u orden inferior) designan el desplazamiento dentro de la página (offset) y el resto (los de más peso, orden superior) el número de página como se ve en la Fig. 5.13.

Si el tamaño del espacio de direcciones es $\mathbf{2^a}$, y el tamaño de la página es $\mathbf{2^n}$ unidades de direccionamiento (bits), el orden superior es **a-n** bits de direcciones lógicas designadas al número de páginas, y el **n** orden inferior bits al desplazamiento de la página. Por ejemplo: para a = 32 bits y n = 12 bits se tendrá una longitud de página de 4 K (2^{12}) y una cantidad de páginas de 1 Mega (2^{20}).

- Se elimina la fragmentación externa ya que, aunque las páginas son continuas, los frames no lo son por lo que genera ubicación dinámica.
- Aparece ***fragmentación interna:*** Memoria perteneciente a una página pero que no es utilizada.

Típicamente la última página de cada zona de memoria ocupada. Si los requerimientos de memoria de un proceso no coinciden con los límites de las páginas, el último frame asignado puede estar parcialmente vacío. En el peor de los casos un proceso necesitará n páginas más una palabra, se le asignarán n + 1 frames, produciendo una fragmentación interna de casi un frame.

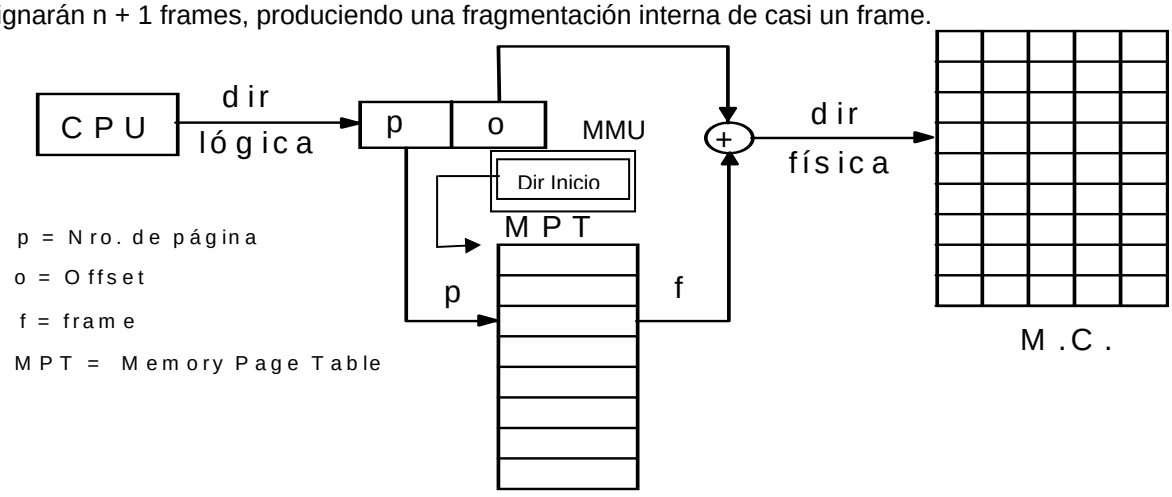


Fig. 5.14 Hardware de paginación

El acceso a la tabla de Páginas lo hace el MMU con la dirección de inicio donde está localizada la tabla en Memoria Central.

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

En la figura 5.14, cada dirección generada por el procesador se divide en dos partes: **o** (desplazamiento), **p** (número de página). El **p** se utiliza como índice en la tabla (MPT) de páginas que contiene la dirección base para cada bloque de la memoria física. La dirección base se combina con el desplazamiento para definir la dirección de memoria física. Esta Localización de una dirección física en Memoria Central lo realiza el hardware del M.M.U., quién calcula a partir de la dirección relativa generada en tiempo de carga (construye la tabla MPT) mediante el siguiente cálculo:

$$f_{(a)} = f(p,o) = p' + o = a,$$

donde p = parte entera de a / λ representa el número de página

o = resto de a /λ corresponde al desplazamiento (offset) dentro de la página y

λ = tamaño de la página.
a = dirección física

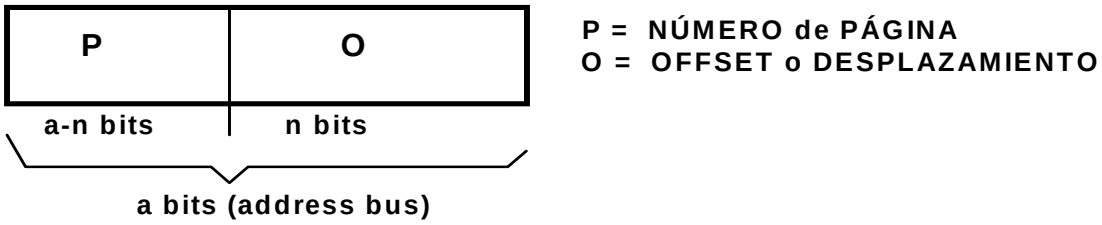


Figura 5.15 Dirección lógica

Observación:
n : corresponde a la longitud máxima de la página (λ normalmente 4 Kpalabras)
a-n: Cantidad máxima de páginas que provee el Sistema.
a : ancho del bus de direccionamiento del Sistema (típicamente 24, 32, 48 o 64 bits)

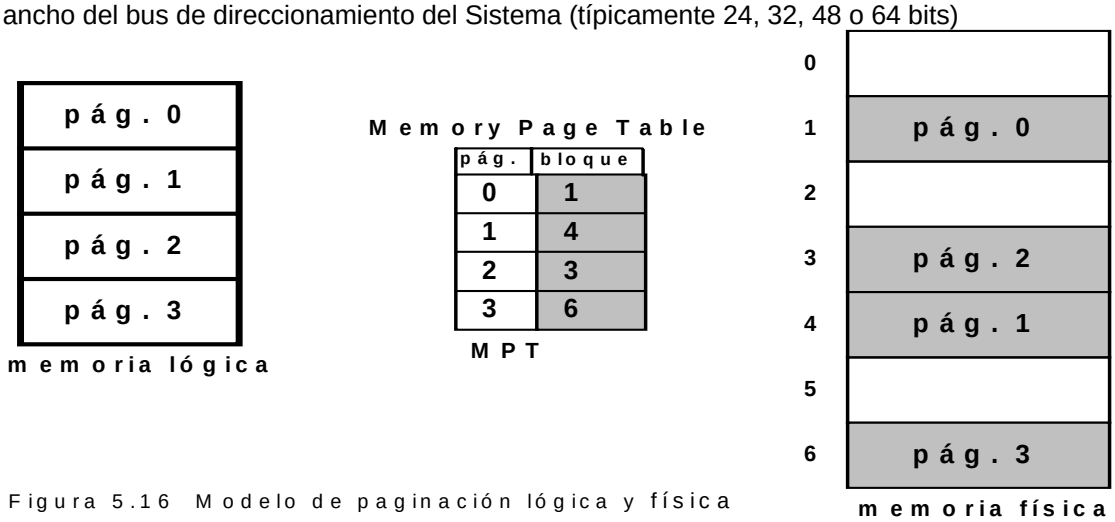


Figura 5.16 Modelo de paginación lógica y física

- ***Funciones del planificador:***

1. Controlar el estado de la Memoria Central. Usa dos tablas:
   - **Mapa de páginas (MPT Memory Page Table)**: Uno por cada proceso activo. Describe los bloques de Memoria Central que están siendo usados por el proceso. Contiene una entrada por cada página del proceso. Existe una por cada trabajo cargado en el sistema.
   - **Mapa de frames o bloques de Memoria Central (MFT Memory Frame Table)**: Existe una sola por Sistema. Describe el estado de cada frame de la memoria central (áreas libres para asignar y las ya asignadas, o sea ocupadas). Esto se puede resolver con un solo bit para indicar el estado de LIBRE o ASIGNADO. Lleva la contabilidad.
2. Determinar qué trabajo recibirá memoria.
3. Asignar memoria central implica cargar el proceso en memoria, inicializar la tabla (MPT) de páginas del proceso y actualizar la MFT ocupando los frames que se le otorga al proceso.
4. Desasignar implica descargar el proceso de memoria: Grabar en el disco las paginas modificadas, actualizar la tabla (MFT) que ocupaban las páginas del proceso pasando a libres los frames que le fuera otorgado al proceso.
5. Proveer una protección a cada página.

La MFT también se la conoce como Tabla del Mapa de Memoria Física compuesta por una entrada para cada Bloque (frame), su estado (libre o asignado) y es una tabla estática única en el sistema. La cantidad de entradas depende de la capacidad máxima de memoria física instalada en el sistema y del tamaño de la página. En el instante de inicialización el S.O. crea esta tabla utilizando la siguiente fórmula:

$$F = m / \lambda$$ donde

F = cantidad de bloques (Frames) que dispondra el Sistema.
m = Tamaño máximo de Memoria física instalada en el sistema.
λ = Tamaño de la Página. Prefijada para el S.O.

Generalmente **m** y λ son potencia de dos por lo que **F** es un número entero.
Cuando se carga un proceso, cuyo tamaño es “t”, el S.O. busca un espacio de “x” bloques libres y lo asigna. Para determinar el valor de “x” se hace:

$$x = [\, t / \lambda \,]$$

x = cantidad de bloques (Frames) que requiere el Proceso
t = Tamaño del Proceso a ser cargado
λ = Tamaño de la Página.
[t/λ] Parte entera

La parte entera **[t/λ]** se toma por exceso en el redondeo del cociente si el resto es distinto a cero.

El proceso de asignación se realiza mediante la siguiente secuencia:

1) buscar "x" cualquiera frames libres de la MFT, Si no hay disponibles se produce una interrupción por no haber espacio suficiente en Memoria (esto se informa al usuario mediante un mensaje en la pantalla), en cambio si hay lugar suficiente pasa al paso 2.
2) Construir la MPT para el proceso que se va a cargar dándole tantas entradas como páginas tenga el trabajo.
3) Cargar el proceso y actualizar las dos tablas simultáneamente asignando en la MPT el bloque a cada página y marcando en la MFT como ocupado.

Pero ¿cómo encontrar "x" frames libres para realizar la asignación en forma eficiente, o sea, con bajo overhead?. La búsqueda en la MFT es poco eficiente por ser una tabla estática. Generalmente se facilita la asignación si se tiene una lista enlazada de bloques libres que se puede suponer distribuido aleatoriamente por toda la memoria. La búsqueda de espacio disponible en memoria central, o sea, la cantidad media de entradas que se consultan (y), se puede calcular mediante:

$$y = x / (1-q)$$

donde:

y = cantidad de entradas buscadas en la MFT para "**x**" bloques que requiere el Proceso
(**1-q**) = es la probabilidad de que un conjunto de bloques está libre.
q = porcentaje de memoria ocupada ÷ 100.
y denota el grado de uso de la memoria.

Ejemplo: Supongamos para una demanda de 10 bloques, la memoria tiene un 60% de sus frames ocupados (q = 0,6), o sea que tiene el 40% libre (1- q) = 0,4. El número de entradas en la MFT que se examinarán será 25.

Esta forma de consultar la tabla estática es relativamente lenta, por lo que se propone una lista enlazada de bloques libres en memoria central. Para la asignación de "x" bloques demandados se buscan las "x" primeras entradas de la lista y se asignan.

En el caso de La tabla estática, la desasignación se realiza modificando el bit de estado, en cambio utilizando una lista enlazada, se incorporan las entradas que se liberan al comienzo de dicha lista. Este proceso (de agregar a la lista por ser un recurso compartido) es seguro ya que durante el tiempo que dura dicho proceso no compromete la situación del estado del recurso memoria en cuanto a su rendimiento.

Como resumen decimos que la dirección lógica está compuesta por el número de página y el desplazamiento. Para convertir la dirección lógica en una dirección física, se deben realizar los siguientes pasos:

1. Extraer el número de página de la dirección lógica. Calcular: dir lógica / λ, toma la parte entera como número de pagina (p) y el resto como desplazamiento (d)
2. Usar el número de página (p) como un índice para encontrar el número de bloque (k) en la tabla de páginas.
3. Calcular la dirección física utilizando la ecuación $a = k * 2^m + d$, donde "a" es la dirección física, "k" es el número de bloque, "m" el número de bits utilizados para el desplazamiento, y "d" el desplazamiento.

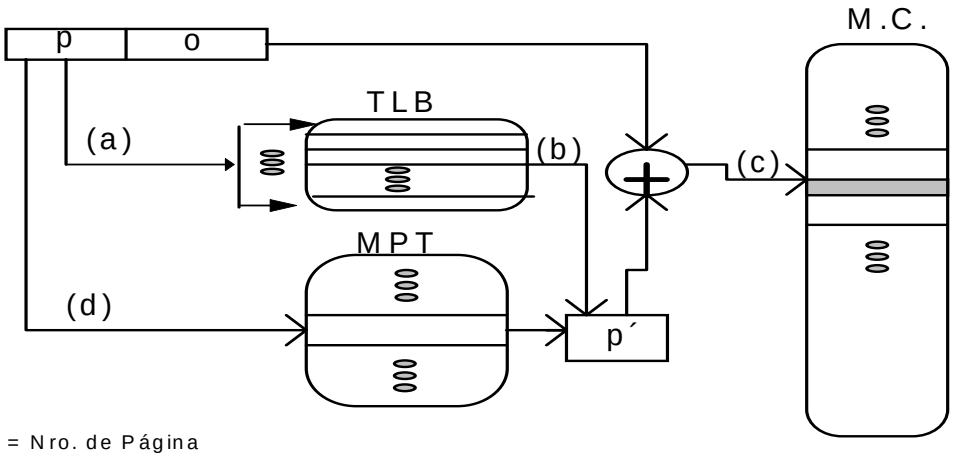
♦ *Funciones implementadas en Hardware:*

a) Mapas de páginas en bancos de *registros de alta velocidad*: Alternativa eficiente, pero muy costosa. La instrucción para cargar o modificar los registros de la tabla de página son privilegiadas, por lo que solo el S.O. puede cambiar el mapa de memoria. El cambio de contexto (context switch) puede ser muy largo debido a la actualización de los contenidos (valores) de estos registros. La utilización de estos registros es satisfactoria si la cantidad de entradas es razonablemente pequeña (Por ej.: 32 entradas o registros).
b) Mapas de páginas en Memoria Central: Registro **memory page table, address base register o page-table base register (PTBR).** La tabla se conserva en la memoria central y un registro PTBR

apunta a la tabla de páginas (esto se esquematiza en la figura 5.23 mas detalladamente). Para cambiar entre tablas de páginas, sólo hay que modificar éste registro, lo que reduce considerablemente el tiempo de cambio de contexto. Si bien el cambio de contexto es corto, se produce un excesivo overhead en los accesos (2 por cada acceso uno para obtener la dirección del bloque y la segunda para la dirección física).

c) Enfoque híbrido: Usa **registros asociativos o Translation Look-aside Buffers (TLB).** La memoria asociativa consiste básicamente en un conjunto de registros que se ubican en la M.M.U. con el objetivo de almacenar allí las referencias de las páginas más usadas con sus correspondientes marcos de páginas. La idea con ello es obtener a partir del número de página el marco de página inmediatamente, sin necesidad de consultar la tabla de páginas. El número de registros pertenecientes a la memoria asociativa es dependiente de la arquitectura. Por lo general son de 16 o 32 registros.
Cada registro contiene el número de la página del proceso y el puntero al comienzo del bloque de memoria central donde está alojada la página. La **M.M.U. (Memory Management Unit)** busca el número de la página en forma simultánea en todos los registros. Cada registro consta de dos partes: una clave (o rótulo - tag) y un valor, y cuando se presenta un elemento a los registros asociativos, se compara simultáneamente con todas las claves. Si alguno la posee, devuelve su dirección y el direccionamiento se realiza. Si no, se hace un acceso a memoria central para buscar la página, y así obtener el número de frame para acceder a memoria.

Si no se encontró en registros asociativos y se accedió a memoria, no sólo se obtiene el frame sino que se añade los números de páginas y frames a los registros asociativos para que puedan localizarse con rapidez en la siguiente referencia. Si los registros del TLB están llenos, el S.O. debe seleccionar uno para actualizar su contenido. Una desventaja es que cuando una nueva tabla de página se selecciona, el TLB debe ser borrado y actualizado para asegurar que la próxima ejecución de un proceso no utilice información errónea, ya que puede haber registros con información desactualizada en el TLB con contenidos de direcciones lógicas válidas, pero direcciones físicas inválidas o incorrectas de los procesos previos.



p = Nro. de Página
o = Desplazamiento dentro de la Página (offset)
TLB = Translation Lookaside Buffer
MPT = Memory Page Table
p´ = Nro. de Bloque (Dirección del Frame)
(a) = primer búsqueda en TLB
(b) = Acierto (TLB Hit)
(c) = Dirección real
(d) = Segunda búsqueda en MPT

Fig. 5.17 Direccionamiento con TLB

El porcentaje de veces que se encuentra un número de página en los registros asociativos se denomina **hit ratio (aciertos).** Si para buscar en éstos registros asociativos se requiere 5 ns y 70 ns para el acceso a memoria, entonces un acceso con correspondencia requiere solo 75 ns cuando el número de página se encuentra en los TLB. Si no lo halla en estos registros (de 5 ns), se acude a la Memory Page Table en memoria para obtener de la tabla el número de frames (70 ns) y luego acceder al lugar de memoria deseado. Esto requerirá un doble acceso (140 ns) en la primer búsqueda, inmediatamente se actualiza la información en los TLB por lo que hace que en el próximo y los sucesivos acceso a esa página será nuevamente de 75 ns. Esto se esquematiza en la Fig. 5.17.

Observación: los tiempos indicados para la memoria RAM de 70 nseg y del TLB de 5 nseg. Son solo indicativos para ejemplificar el uso de esta memoria asociativa. Hoy día la tecnología ha superado la barrera de 10 nseg y 1 nseg para cada uno de ellos.

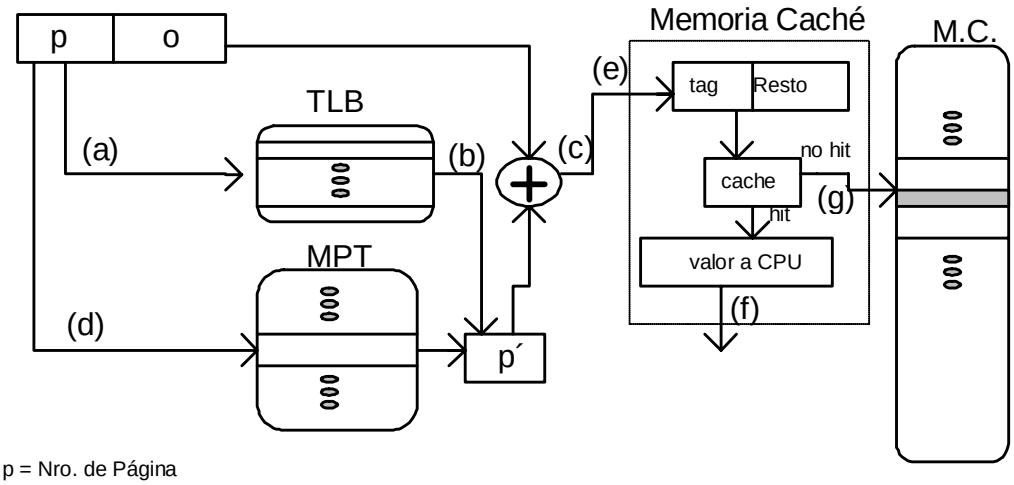
Los M.M.U. incorporados a los chips de procesadores suelen usar entre 16 y 512 registros asociativos por lo que los fabricantes suelen garantizar un hit ratio de acuerdo a la cantidad de TLB entre 80% a 98%.

Ejemplo: 68030 con 22 TLB obtiene un 90% de hit ratio (aproximadamente).
80486 con 32 TLB obtiene un 98% de hit ratio (aproximadamente).

La búsqueda es muy rápida pero el hardware es muy costoso.

Como el procesador es mas rápido que la memoria central deberá esperar varios ciclos de reloj hasta ver satisfecho sus requerimientos. Para evitar grandes esperas se recurre a las memoria caché. El esquema de la figura 5.18 ejemplifica la búsqueda final de la dirección física.

Direccionamiento con TLB y Caché:



p = Nro. de Página
o = Desplazamiento dentro de la Página (offset)
TLB = Translation Lookaside Buffer
MPT = Memory Page Table
p´ = Nro de Bloque (Dirección del Frame)

(a) = primer búsqueda en TLB
(b) = Acierto (TLB Hit)
(c) = Dirección búsqueda en caché
(d) = Segunda búsqueda en MPT
(e) = Operación en caché
(f) = Búsqueda exitosa, el Valor a CPU
(g) = Búsqueda fallida, acceso a M.C.

Fig. 5.18 Direccionamiento con TLB y Caché

En lugar de acceder a memoria busca la información en la memoria caché. Si tiene éxito, la información va directamente a la CPU, demorándose solo el acceso de la caché de unos 5 a 10 ns en lugar de los 70 ns de la RAM. Si no esta en la caché, entonces va a la RAM a buscar la información y actualizar la caché con la página nueva. En el próximo acceso probablemente ya tendrá éxito la búsqueda en la caché.

Algoritmo con TLB

El algoritmo utilizando TLB se observa en la Fig. 5.19. Es muy sencillo y eficiente. Generalmente se lo cablea en hardware (M.M.U.) para que sea más rápido en su ejecución.

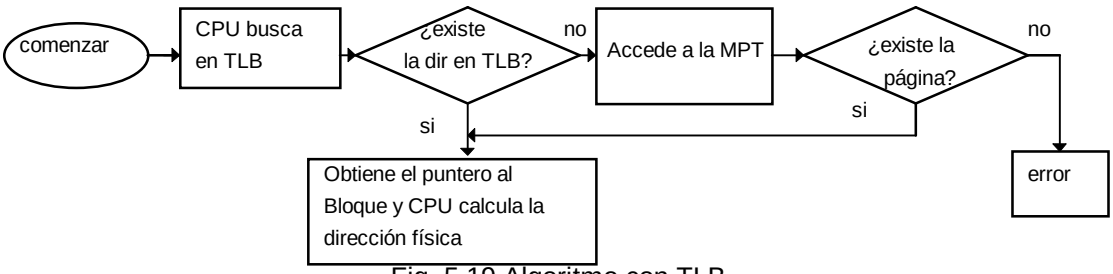


Fig. 5.19 Algoritmo con TLB

Otra forma de generar las direcciones es utilizar una tabla de página invertida adicional como se muestra en la figura 5.20.

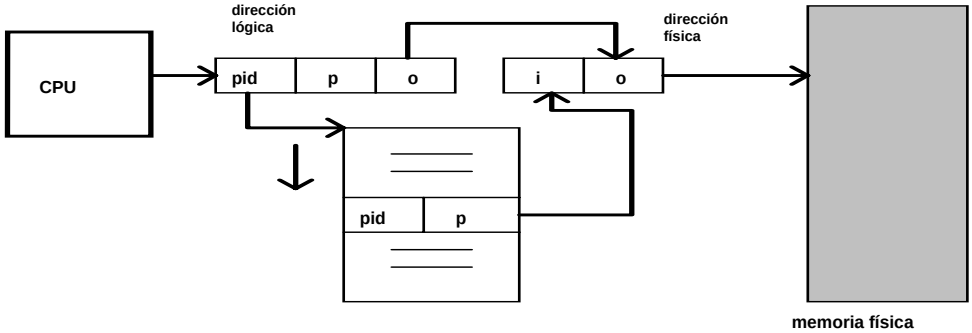


Figura 5.20 Tabla de páginas invertida

Comparación entre las Tablas de páginas

TABLA DE PÁGINAS (MPT)	TABLA DE PÁGINAS INVERTIDA
Una tabla asociada a cada proceso	Una tabla en todo el sistema
Contiene una entrada por cada página que usa el proceso.	Contiene una entrada para cada página real (frame) de memoria. Con esa entrada se asocia la dirección virtual de la página almacenada en esa localidad.
Los procesos hacen referencia a las páginas por medio de direcciones virtuales de las páginas. Se calcula a través del número de página y el desplazamiento. El S.O. debe traducir esta referencia a una dirección de memoria física.	Se presenta al subsista un parte de la dirección virtual <id-proceso, número-pág>, luego se recorre la tabla para buscar una coincidencia, si la hay se genera la dirección física, sino hay un page fault o un error.
Ventajas: - Búsqueda en la tabla es rápida, dado que está ordenada por las páginas. - Permite memoria compartida.	*Ventajas:* - Reduce cantidad de memoria para almacenar cada tabla de páginas. - Al conservar información sobre qué página de memoria virtual se almacena en cada frame, reducen la cantidad de memoria física necesaria para almacenar esta información.
Desventajas: - Tamaño grande de las tablas de páginas. - Se consumen grandes cantidades de memoria física, sólo para controlar la forma en que se utiliza el resto de la memoria física.	*Desventajas:* - Está ordenada por dirección física, pero las búsquedas se efectúan por direcciones virtuales, recorre toda la tabla para encontrar una coincidencia. Aumenta el tiempo necesario para buscar en la tabla cuando hay una referencia a una página. - Las tablas de dispersión que limita la búsqueda a una o más entradas de la tabla, añade una referencia a memoria. - No contiene información completa sobre el espacio de direcciones lógicas de un proceso. Por ejemplo no muestra en que lugar del almacenamiento secundario se encuentra una página que no está en memoria. Para esto mantiene tabla de páginas externas (como las tradicionales) Como solo se las necesita cuando hay una falla de página, se descargan y reincorporan a memoria según se las necesite. - No permite memoria compartida. Un frame no puede tener dos (o más direcciones virtuales compartidas).

Tabla 5.1 comparación entre MPT y Tabla de página invertida

Páginas compartidas

Sólo se pueden compartir páginas cuando dicha página esta programada como código reentrante (o puro, que no se modifica a sí mismo). Nunca cambia durante la ejecución, de manera que dos o más procesos pueden ejecutar el mismo código al mismo tiempo.

Consiste en apuntar las tablas de páginas de todos los procesos a los mismos frames.

Cada proceso tiene su propia copia de los registros y su almacenamiento para contener los datos durante la ejecución. Los datos variarán para cada proceso. Por ejemplo, en este método, si se necesita un editor de textos de 150 kiBy de código y 50 kiBy de espacio de datos, no se necesitarían 8000 kiBy para apoyar a 40 usuarios, sino que, únicamente, se requiere una copia del editor (150 kiBy) más 40 copias del espacio de datos de 50 kiBy para cada usuario, entonces se requerirá en total 2150 kiBy solamente.

Este forma de compartir la memoria entre procesos es similar al de compartir hilos (threads) en el espacio de direcciones de una tarea.

Ventajas:

1. - Especial importancia en un entorno de tiempo compartido.
2. - Ahorro considerable del espacio de memoria.
3. - Se pueden compartir programas de uso frecuente: compiladores, sistemas de bases de datos, etc.

Desventajas:

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

- **Protección**

- Se usan Bits de protección asociados con cada frame (R, W, X, que significa Read, Write y Execute respectivamente). Normalmente estos bits se conservan en la tabla de páginas. Al mismo tiempo que se calcula la dirección física, pueden consultarse los bits de protección. Por ejemplo, un intento de escribir en una página de sólo lectura ocasiona una interrupción al S.O. (violación de protección de memoria).
- Un bit es adherido a cada entrada de la tabla de página: **bit válido - inválido**. El S.O. pone este bit para cada página, para permitir o no el acceso a la página. Cuando el bit es "*válido*" este valor indica que la página asociada está en el espacio de direcciones lógicas del proceso, y es entonces una página legal. Si es "*inválido*" no está en el espacio de direcciones lógicas y es ilegal. (válido puede ser 1 e inválidos = 0)

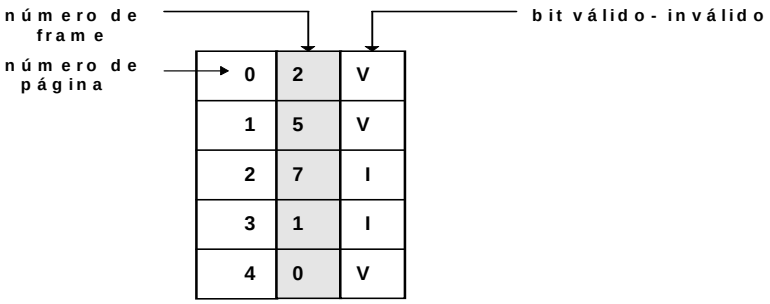


Figura 5.21 Bit válido-inválido en la tabla de páginas MPT

- Es poco común que un proceso utilice todo su intervalo de direcciones, por lo tanto es un derroche crear una tabla con registros para cada página del intervalo de direcciones. La mayor parte de esta tabla no se utilizaría, pero ocuparía un valioso lugar en memoria. Algunos sistemas ofrecen hardware que indica el tamaño de la tabla de página a través de un **Page Table Length Register(PTLR)** para que el proceso no pueda direccionar más allá de un cierto límite.

Bit de Modificación de página:

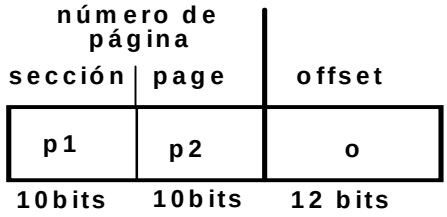


Figura 5.22 Direccionamiento lógico de 32 bits

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

Paginación de multinivel:

Los sistemas más modernos de computación soportan un gran espacio de direcciones lógicas (2^{32} a 2^{64}), en tal entorno la tabla de página puede volverse excesivamente grande.

Un camino es usar el esquema de dos niveles de paginación, en el cual la tabla de página también es paginada. Un ejemplo lo vemos en la figura 5.22 en una máquina de 32 bits, con un tamaño de página de 4 KiBytes, una dirección lógica es dividida en un número de páginas de 20 bits y un desplazamiento de 12 bits. A la vez el número de página es dividido en 10 bit para el número de página (1024 entradas) y 10 bits para la página de desplazamiento o sección (1024 entradas) donde p1 es un indexado dentro de la tabla de página externa y p2 es un desplazamiento junto con la página de la tabla de página como si fuera una MPT común.

También se puede usar un esquema de paginación de tres niveles como el indicado en la Fig. 5.23, donde la tabla de páginas externas también es paginada (resultando un segundo nivel de página externa). El próximo esquema será de cuatro niveles de paginación, donde el segundo nivel de tabla de páginas externa también es paginada. Esto depende de la tecnología del hardware para integrar las funciones del M.M.U. y hacerlo eficiente.

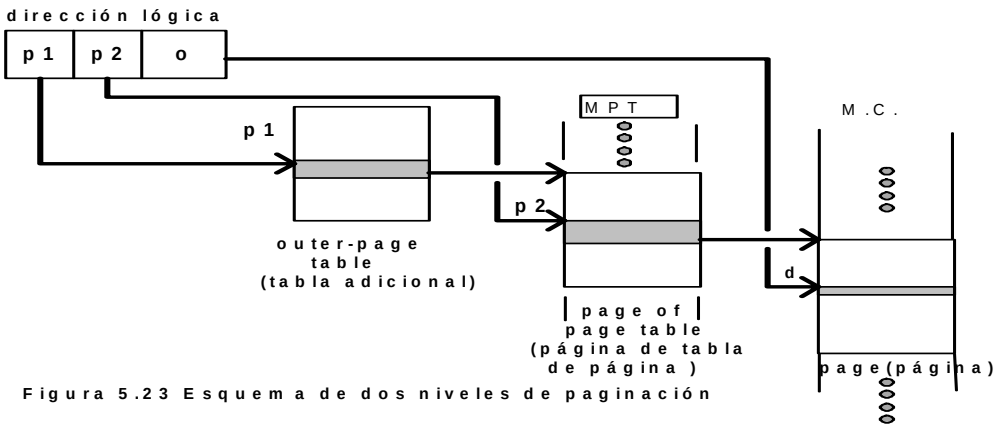


Figura 5.23 Esquema de dos niveles de paginación

Este esquema busca economizar bits en el direccionamiento y en el tamaño de las tablas. Por ejemplo: con un direccionamiento de 32 bits, se usan 10 bits para p1, 10 bits para p2 y 12 bits para el tamaño de las páginas, permitirían programas de 4 MeBy de longitud con solamente manejar p2 y offset. Esta longitud es apta para que la mayoría de programas se carguen con este espacio de direccionamiento. El campo p1 permanecería fijo en cero.

Este esquema busca economizar bits en el direccionamiento y en el tamaño de las tablas. Por ejemplo: con un direccionamiento de 32 bits, se usan 10 bits para p1, 10 bits para p2 y 12 bits para el tamaño de las páginas, permitirían programas de 4 MeBy de longitud con solamente manejar p2 y offset. Esta longitud es apta para que la mayoría de programas se carguen con este espacio de direccionamiento. El campo p1 permanecería fijo en cero.

Ventajas de la paginación

1. Elimina el problema de fragmentación externa.
2. Mayor grado de multiprogramación.
3. No necesita compactación.
4. Permite que la memoria de un proceso no sea contigua, y que a un proceso se le asigne memoria física donde quiera que ésta esté disponible. Evita el gran problema de acomodar trozos de memoria de tamaño variable en el almacenamiento auxiliar.

Desventajas

1. Existe Fragmentación interna.
2. Hardware mas caro.
3. Menor velocidad debido a la constante traducción de direcciones lógicas en físicas.
4. Gasta más memoria central utilizado con tablas.
5. El PCB de cada proceso crece, debe almacenar los contenidos del espacio de direccionamiento.
6. Espacio de memoria de un proceso debe ser menor o igual que el de memoria central libre para alojarlo (esta es una importante restricción)

5.3.5. Segmentación Simple

Un aspecto importante de la administración de memoria que se volvió inevitable con la paginación es la separación de la visión del usuario de la memoria y la actual memoria física, la cual no es la misma; sino que la visión del usuario se “mapea” sobre la memoria física. El mapeo permite la diferenciación entre la memoria lógica y la memoria física.

Surge como un acercamiento a la forma en que los usuarios ven a la memoria central y en particular con el auge de la programación estructurada.

Existe un consenso general de que el programador o usuario de un sistema no considera la memoria como un arreglo lineal de palabras, sino que la piensa como un conjunto de segmentos de tamaño variable, sin ningún orden en especial.

La Segmentación es un esquema de administración de memoria que apoya la perspectiva que el usuario tiene de la memoria. Un espacio de direcciones lógicas se compone de un conjunto de segmentos, cada uno de los cuales tiene un nombre y una longitud. Las direcciones especifican el nombre del segmento y el desplazamiento dentro de él, de manera que el usuario especifica cada dirección con dos cantidades: el nombre del segmento y un desplazamiento.

Para simplificar la implementación, los segmentos se enumeran y se referencian por su número, mas bien que por el nombre. Así, una dirección lógica consiste de una tupla:

Número de segmento	desplazamiento

Normalmente, el programa del usuario se compila y el compilador automáticamente construye segmentos que reflejan el programa de entrada.

Aunque el usuario puede ahora referirse a objetos en el programa por una dirección bidimencional, la memoria física es una secuencia de bytes unidimencional. Por lo tanto, se define una implementación para mapear direcciones definidas por el usuario bidimencionales dentro de una dirección física unidimencional. Este mapeo es efectuado por una ***tabla de segmento (Segment Memory Table SMT)***.

El programa dividido en sus distintos segmentos (ejemplo subrutinas) se pueden cargar en distintos espacios de direccionamiento no contiguo.

Cada entrada de la tabla tiene una base del segmento y un límite del segmento. La base del segmento contiene la dirección física de comienzo donde el segmento reside en la memoria, en tanto que el límite del segmento especifica la longitud del segmento o sea su tamaño.

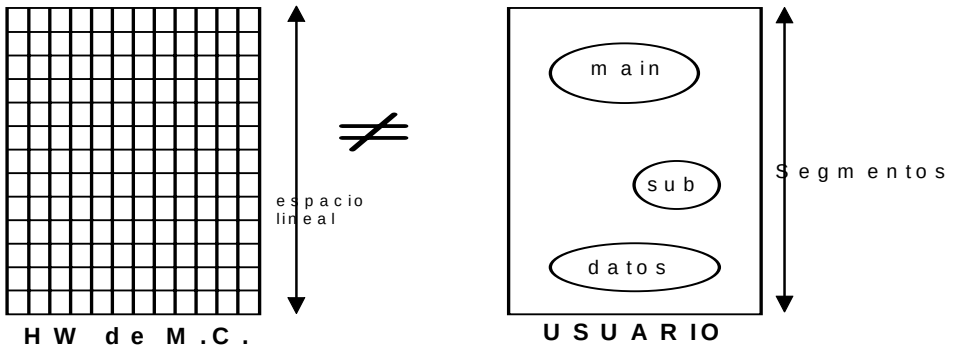


Fig. 5.24 comparación entre segmentos lógicos y hardware

El número de segmento se usa como un índice dentro de la tabla de segmentos. El desplazamiento de la dirección lógica debe ser entre 0 y el límite del segmento. Si no es así se bifurca al S.O. para que lo trate como un error en el direccionamiento considerándolo ilegal. Si el desplazamiento es legal, éste se suma a la base del segmento para producir la dirección en la memoria física de la posición requerida. La tabla de segmento es así esencialmente un arreglo de pares de registros base y límite.

- **Segmento**:

Se define a un segmento como el conjunto lógicamente relacionado de datos o códigos, generado por el usuario. Su tamaño es arbitrario y definido por el usuario según el propósito del segmento en el programa.

Entonces cada segmento tiene asociado un nombre o número y una longitud.
El usuario especifica la dirección en dos partes: Nombre de segmento y desplazamiento, (offset, dentro del mismo).

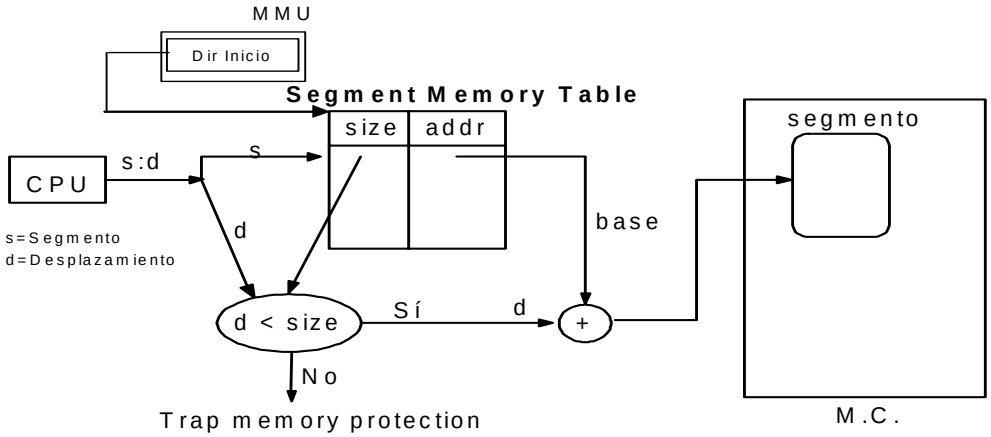


Fig. 5.25 Direccionamiento en segmentación

Los segmentos son numerados generalmente en el compile time y el compilador genera los segmentos de acuerdo al programa. La Tabla de Segmentos (SMT = Segment Memory Table) se genera en tiempo de traducción. Existe una Tabla por Trabajo.

Esta tabla permite implantar la correspondencia entre direcciones bidimensionales definidas por el usuario y direcciones físicas unidimensionales.
Una dirección lógica consiste en dos partes: **s** y **d**.
s se utiliza como índice en la tabla de segmentos. Cada entrada de la tabla de segmentos tiene una ***base*** y un ***límite*** para el segmento.

El **d** de la dirección lógica debe estar entre 0 y el límite del segmento, si no se genera una trampa al S.O. Si el **d** es válido, se suma a la base del segmento para producir la dirección en memoria física de la palabra deseada. La tabla de segmentos en esencia es un arreglo de pares de registros base y límite.

80286: 16 bits para especificar el segmento $\Rightarrow 2^{16}$ entradas = 64k segmentos. Desplazamientos de 24 bits $\Rightarrow$ cada segmento = 2^{24} Bytes = 16 MBy
80386: Desplazamientos de 32 bits $\Rightarrow$ cada segmento = 2^{32} Bytes = 4 GiBy
80486: Idem

Soporte de Hardware para la segmentación:

Segmentación es similar a particiones (dinámicas), pero un programa se divide en varios segmentos. Ubicación de la SMT (Segment Memory Table) o Tabla de Segmentos presenta el mismo problema que en paginación con el acceso. Cada direccionamiento lógico debe ser traducido mediante un acceso a la tabla. La forma de llevar esta tabla tendrá una gran influencia en la velocidad del procesamiento. Se presentan tres soluciones:

1. Registros dedicados en la M.M.U. (rápido y caro). Para ahorrar tiempo, la suma a la base y la comparación con el límite pueden efectuarse simultáneamente.
2. Memoria RAM (lento y barato): Se incorporan dos registros especiales en el procesador para saber la ubicación y longitud de la tabla de segmentos; SMTBR (Segment Memory Table Base Register), que apunta a la tabla de segmentos y SMTLR (Segment Memory Table Length Register), ya que el número de segmentos que utiliza un programa puede variar considerablemente. Para una dirección lógica (s, d), primero se comprueba si s < STLR, luego se suma (s + STBR), lo que da como resultado la dirección en memoria de la entrada de la tabla de segmentos. Y luego se calcula la dirección física como se ha explicado anteriormente. La desventaja es que requiere dos referencias a memoria por cada dirección lógica.
3. Enfoque híbrido: Se construye un caché de segmentos de la misma manera que en la paginación. Se emplea un conjunto de registros asociativos para que contengan las entradas de la SMT usadas más recientemente.

- ***Segmentos compartidos:***

Otra ventaja de la segmentación involucra el compartir códigos o datos. Cada proceso tiene una tabla de segmento asociada con su PCB, a quien el dispatcher usa para definir los contenidos de la tabla de segmento cuando a éste proceso se le da la CPU. Los segmentos son compartidos cuando las entradas en las tablas de segmento de dos procesos diferentes apuntan a las mismas locaciones físicas. El compartir ocurre a nivel de segmentos. Así, cualquier información puede ser compartida si ésta se definió previamente para ese segmento.

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

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

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

Protección de Segmentos compartidos:

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

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

Los Segmentos se pueden proteger (contra Lecturas, escrituras o ejecución) y especificar su tipo (R, W, X, etc.) agregando bits a la SMT. Esto es muy caro por lo que se propone agregar un puntero a una Tabla de Protección como se indica en la Fig. 5.26, en donde cada entrada tiene asociado los procesos y sus permisos.

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

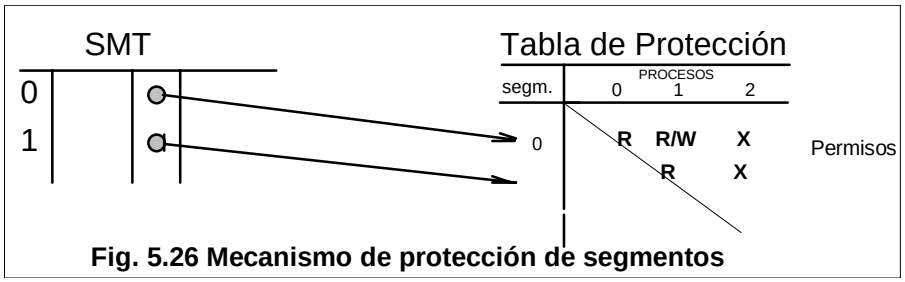


Fig. 5.26 Mecanismo de protección de segmentos

- **Fragmentación :**

Problema: en este sistema de administración reaparece fragmentación externa. Un segmento debe ser una unidad o sea, estar contiguo en memoria central.

El long term scheduler debe encontrar y asignar memoria para todos los segmentos del programa de usuario, ya que debe ser cargado íntegramente en la memoria central. Esta situación es similar a paginación excepto que los segmentos son de longitud variable; en tanto que las páginas son todas del mismo tamaño. Así, como con la técnica de particiones de tamaño variable, la asignación de memoria es un problema del tipo de asignación de almacenamiento dinámico, usualmente resuelto con un algoritmo first fit o best fit.

La segmentación puede causar fragmentación externa cuando todos los bloques de memoria libres son demasiado pequeños para acomodar un segmento. En ese caso, el proceso puede tener que separar hasta que se disponga de más memoria, o puede usar la compactación para crear una partición más amplia. Dado que la segmentación es por naturaleza un algoritmo de reubicación dinámica, se puede compactar la memoria siempre y cuando se quiera.

Si el planificador de la CPU debe esperar por un proceso debido a problemas de asignación de memoria, éste puede recorrer la cola buscando uno más pequeño o un proceso de prioridad más baja para correr.

La seriedad del problema de la fragmentación externa depende principalmente del tamaño del segmento promedio. Generalmente, si éste promedio es pequeño, la fragmentación será también pequeña; dado que los segmentos individuales son más grandes que el proceso, ellos son más probables de encontrar en los bloques de memoria disponible.

Solución:

- Compactación implica tiempos largos (Overhead por compactación).
- Paginar los segmentos individuales implica Fragmentación interna lo que produce overhead en el direccionamiento.
- Swapping: sacar de memoria los segmentos inactivos para poder cargar uno nuevo.
- Sólo cargar en memoria central algunos segmentos y después ir cargando a medida que se vayan necesitando (uso de memoria virtual).
- Cuando se referencia un segmento que no está en memoria central se produce una interrupción por segmentación por lo que el S.O. busca lugar en memoria para el nuevo segmento.

Otras posibilidades sin mayores ventajas:

- Se puede definir cada proceso como un segmento, pero sería reducirla a un esquema de particiones variables.
- Cada palabra puede colocarse en su propio segmento y relocalizarse por separado. Esto eliminaría la fragmentación externa, sin embargo, cada palabra necesitaría un registro base para su relocalización, duplicando la utilización de memoria.
- Pequeños segmentos de tamaño fijo como en la paginación.

Modificación de un Segmento en Memoria Central.

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

5.3.6. Manejo de memoria con Buddy System

Buddy system (Knuth,1973; Knowlton, 1965) es un algoritmo de manejo de memoria que, aprovechando que las computadoras usan números binarios para direccionar, hace que sea muy rápida la unión de huecos adyacentes cuando un proceso termina o es llevado a disco (swapped-out).

Funciona de la siguiente forma:

El administrador de memoria mantiene una lista de bloques libres de tamaño 1, 2, 4, 8, 16, etc., bytes hasta el tamaño de toda la memoria. Con una memoria de 1 MeBy, por ejemplo, se necesitan 21 listas (desde 1 byte hasta 1 Megabyte). Inicialmente, toda la memoria está libre, y la lista tiene una sola entrada que contiene un hueco de 1 MeBy las otras listas están vacías. El primer renglón de la siguiente figura muestra la configuración inicial.

RENGLÓN 1: La memoria se encuentra vacía.
RENGLÓN 2: Veamos como funciona Buddy system si se carga un proceso ("A") de 70 KiBy (las letras en la Fig. 5.27 representan los bloques de memoria asignados y los renglones los huecos). Como en la lista de huecos solo hay potencias de 2, se requieren 128 KiBy (la menor potencia de 2 lo suficientemente grande). Pero no hay ningún bloque de 128 KiBy. , ni de 256 KiBy o de 512 KiBy. Entonces, el hueco de 1 MeBy. es dividido en dos bloques de 512 KiBy "compañeros" (Buddies), uno en la dirección 0 y el otro en la dirección 512 KiBy.
El que comienza en la dirección 0 es separado en dos bloques compañeros de 256 KiBy, uno en la dirección 0 , y el otro en la dirección 256 KiBy . Luego el de menor dirección se divide en dos bloques de 128 KiBy, y finalmente, se le asigna al proceso el hueco de la dirección 0 (siempre la menor dirección)..

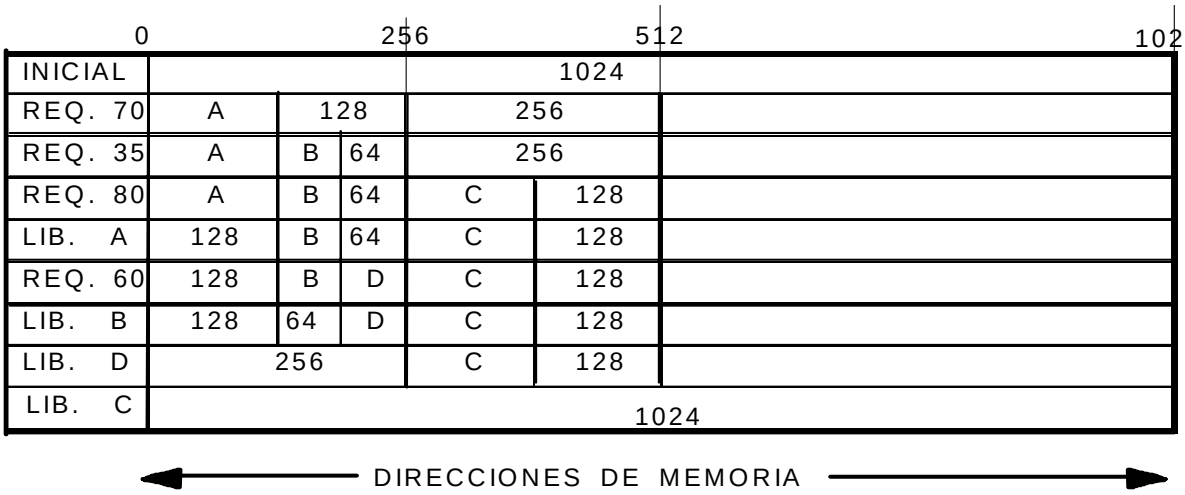


Figura 5.27 Administración con Buddy System

R

ENGLÓN 3: Luego se carga un proceso ("B") de 35 KiBy , en consecuencia debemos buscar un hueco de 64 KiBy , y vemos que no hay ningún bloque de 64 KiBy disponible, así que dividimos el bloque de 128 KiBy en dos bloques compañeros de 64 KBy, uno en la dirección 128 KiBy , y el otro en 192 KiBy. Se le asigna al proceso el bloque en 128 KiBy.
RENGLÓN 4: El tercer requerimiento es de 80 KiBy, y es asignado al primer bloque libre de 128 KiBy.
RENGLÓN 5: Veamos que sucede cuando se devuelve un bloque (se libera la memoria): Supongamos que el proceso "A" finaliza, en ese momento se libera un bloque de 128 KiBy (de los cuales solo 70 KiBy están ocupados). Como no tiene ningún compañero con quien unirse para formar un hueco mayor, pasa a la lista de bloques libres de 128 KiBy.
RENGLÓN 6: Ahora se necesita un bloque de 60 KiBy , así que se verifica si hay algún bloque suficientemente grande. El bloque de 64 KiBy ubicado en la dirección 192 KiBy esta libre, así que se le asigna al proceso.
RENGLÓN 7: Luego se libera el espacio ocupado por el proceso "B". En este momento, tenemos un bloque libre de 128 KiBy en 0 , y otro de 64 KiBy en la dirección 128 KiBy (el que acaba de liberar el proceso "B"). Todavía no es posible hacer ninguna unión de bloques (recordar que el tamaño de los bloques libres siempre debe ser potencia de 2) .
RENGLÓN 8: Cuando el proceso "D" libera la memoria, podemos unir los bloques y construir un hueco de 256 KiBy.
RENGLÓN 9: Finalmente, cuando el proceso "C" libera la memoria, volviendo a la configuración inicial de un solo hueco de 1 MeBy.

CONCLUSIÓN: La ventaja de los BUDDY SYSTEMS es que cuando se libera un bloque de $\mathbf{2}^k$, el administrador de memoria solo tiene que buscar en la lista de huecos de $\mathbf{2}^k$ para ver si es posible hacer alguna unión de bloques (merge), en cambio otros algoritmos que permiten que los bloques se dividan de forma arbitraria deben buscar en toda la lista.

Desafortunadamente, el precio que hay que pagar por esto es una gran ineficiencia en términos de aprovechamiento de memoria. El problema surge del hecho que todas las peticiones deben ser

aproximadas a potencias de 2 sin pasarse (a un proceso de 35 KiBy se le debe asignar un bloque de 64 KiBy, los 29 KiBy extra son desperdiciados).

Resumen de las principales características de la gestión de la memoria sin swapping.

Técnica de administración	Características	Ventajas	Desventajas
Sin S.O.	➢ No hay S.O. ➢ La traducción del código binario se realiza manualmente. Se cargan los datos y el programa mediante **Switches** o manualmente mediante un teclado hexadecimal y se arranca la ejecución poniendo en el **Program Counter** la primer dirección del programa a ejecutar. ➢ El manejo de los recursos lo hace el programa ya que es posible reemplazar los servicios brindados por el S.O., mediante la programación en lenguaje de máquina por el usuario. Luego no se necesita el soporte del S.O. ➢ El usuario tiene toda la M.C. disponible, todas las acciones de control y toda la responsabilidad.	1. Máxima flexibilidad. 2. Máxima simplicidad. 3. Mínimo costo.	1. No existen servicios para interrupciones y errores. 2. Baja productividad para el usuario y el hardware, su programación es engorrosa y compleja. 3. Imposible la construcción de programas largos o grandes y la multiprogramación.
Contigua simple o Monitor residente	➢ Es monoprogramado. ➢ Se divide la memoria en 3 áreas. ➢ Comúnmente, se pone el S.O. en la zona baja de memoria, cerca del vector de interrupción y el programa a continuación separado por un registro barrera como protección.	- Simplicidad. Un S.O que use este plan puede requerir apenas un KiBy, en su totalidad comparados con los 256 KBy o más que necesitan los más complejos. - Facilidad de comprensión y uso. - Es muy simple de implementar. - El S.O es sencillo, pequeño y produce poco overhead de direccionamiento. - El Hardware de protección y direccionamiento es poco costoso y complejo.	1. a memoria no se utiliza al máximo. 2. na parte de la memoria no se usa en absoluto y está asignada al programa de usuario. 3. n la mayoría de los sistemas con canales o DMA, hay ocasiones en que la CPU no está usando activamente alguna porción de la memoria. 4. lgunas porciones de memoria del espacio de dirección del usuario contienen información que jamás se accesa o usa. 5. alta de flexibilidad. 6. o permite la multiprogramación.
Particionada Fija	➢ La especificación de la partición la puede designar el usuario o puede incorporarse en el S.O. ➢ Cada paso de trabajo proporcionado por un usuario debe especificar la máxima cantidad de memoria necesaria. De esta manera se asigna una partición de tamaño suficiente. ➢ El algoritmo del S.O que maneja la	1. Sencillo de implementar. 2. poco overhead del S.O.	1. Empleo ineficiente de la memoria debido a fragmentos sin usar 2. El numero de procesos es fijo.

	asignación y desasignación es simple. ➢ Se divide la memoria en particiones de tamaño fijo. ➢ Cada partición contiene un proceso. ➢ La elección de un proceso situado en la cola de trabajos se asigna a una partición libre. ➢ Usado originalmente por **OS/360 MFT** (**Multiprogramming with a Fixed Number of Tasks**) con una cola de Jobs por cada partición, que luego se reemplazó por una sola cola. ➢ El nivel de multiprogramación está limitado por el número de particiones, cuando una de las particiones está libre se selecciona un proceso de la cola de entrada y se carga en la partición libre, cuando el proceso termina, la partición está disponible para otro. ➢ Esta técnica de partición estática es apropiada cuando los tamaños y frecuencias de los trabajos se conocen, pero se puede desperdiciar gran cantidad de memoria si no se tiene conocimientos de los mismos. ➢ Los tamaños de las particiones son fijos y no cambian durante la ejecución.		
Particionada Variable	➢ Las particiones son creadas durante la ejecución de los procesos para ajustarse a sus necesidades, de ésta manera se hace corresponder los tamaños de las particiones con los de los trabajos. ➢ La asignación de partición se crea dinámicamente en el instante de carga del proceso ➢ Las tablas deben constituirse con entradas para cada área libre y cada partición asignada, donde se especifica el **tamaño**, **posición y restricciones de acceso** a cada partición. ➢ Se usan dos tablas distintas, una para las **áreas asignadas** y otra para mantener el **estado** de las **áreas no asignadas o libres**. ➢ Inicialmente toda la memoria está disponible para los procesos del usuario y se lo considera como un bloque o **hole** (**hueco**) de memoria disponible. ➢ A la llegada de un proceso se busca un bloque de tamaño suficiente para ese proceso, al hallarse sólo se asigna el la cantidad de memoria necesaria. ➢ El tamaño está determinado por la longitud del proceso a ejecutar.	1. Aumenta la multi-programación. 2. se usa mas eficientemente la memoria 3. los algoritmos son simples y fáciles de implementar. 4. Hardware poco costoso.	1. hay fragmentación externa 2. se requiere compactar 3. overhead grande 4. los espacios requeridos por los procesos debe ser contiguo. 5. aumenta el uso de memoria por el S.O. debido a las tablas. 6. El Hardware de direccionamiento y reubicación hace más lento al sistema. 7. Se necesita conocer la longitud del programa a priori para la asignación.
Paginación Pura	➢ La memoria se divide en un conjunto de frames de igual tamaño ➢ Cada proceso se divide en paginas del mismo tamaño que el frame. ➢ Un proceso se carga con todas las paginas en memoria en frames libres no contiguos.	1. Elimina el problema de fragmentación externa. 2. Mayor grado de multiprogramación. 3. No necesita compactación. 4. Permite que la memoria	1. Existe Fragmentación interna. 2. Hardware mas caro. 3. Menor velocidad debido a la constante traducción de direcciones lógicas en físicas. 4. Gasta mas memoria central con tablas. 5. El PCB de cada

		de un proceso no sea contigua, y que a un proceso se le asigne memoria física donde quiera que ésta esté disponible. Evita el gran problema de acomodar trozos de memoria de tamaño variable en el almacenamiento auxiliar.	proceso crece. 6. Espacio de memoria de un proceso debe ser menor o igual que el de memoria central libre para alojarlo (esta es una importante restricción)
Segmentación simple	➢ Cada proceso se divide lógicamente en un conjunto de segmentos ➢ Un proceso se carga en particiones que no necesitan ser contiguas. ➢ La dirección lógica se divide en Nro. de segmento y un desplazamiento.	1. separación de la visión del usuario de la memoria y la actual memoria física 2. protección de los segmentos por hardware 3. facilidad para compartir códigos o datos	1. Overhead por compactación. 2. Paginar los segmentos individuales produce Fragmentación Interna que implica Overhead en el direccionamiento. 3. El Swapping saca de memoria los segmentos inactivos para cargar uno nuevo. 4. Se carga en MC algunos segmentos y después se carga acorde a la necesidad. 5. Al referenciarse un segmento que no está en MC se produce una interrupción por segmentación donde el SO busca lugar en MC para el nuevo segmento.
Buddy System	➢ es un algoritmo de manejo de memoria que, aprovechando que las computadoras usan números binarios para direccionar, hace que sea muy rápida la unión de huecos adyacentes cuando un proceso termina o es llevado a disco	1. Hardware sencillo y poco overhead en la funciones de liberar o asignar espacio. 2. Muy rápido.	1. Mal aprovechamiento de la memoria por los fragmentos.

Tabla 5.2. Resumen de las principales características de la gestión de la memoria sin swapping**.**

Implicancia de las técnicas de administración de memoria sin swapping.

Comparando la paginación simple y segmentación simple, por un lado, con las particiones fijas y dinámicas, en el otro, vemos que la evolución histórica ha aportado una base de conocimientos fundamental en la administración de memoria. Se destacan dos características de paginación y segmentación en esta evolución:

1. Todas las referencias a memoria dentro de un proceso son direcciones lógicas que se traducen dinámicamente en las direcciones físicas en tiempo de ejecución. Esto significa que un proceso puede intercambiarse en y fuera de memoria central tal que ocupe regiones diferentes de memoria central en los momentos diferentes durante el curso de ejecución.
2. Un proceso puede descomponerse en varios objetos (páginas o segmentos) y estos objetos no necesitan estar alojadas en forma contigua en la memoria central durante la ejecución. La

combinación de traducción de direcciones dinámicas en tiempo de ejecución y el uso de una página o tabla del segmento es lo que permite que esto se produzca.

Si el segmento o página, que contiene la próxima instrucción para traerse y el objeto que contiene la ubicación del próximo dato para en ser accedido, están en la memoria central, entonces no hay inconveniente para proseguir la ejecución, pero, se presenta un problema cuando no se encuentra localizado en la memoria central. Entonces se produce un trap y el programa abortaría su ejecución. Esto generó que se modificara esta excepción mediante el Hardware (First Level Interrupt handler) para que no abortara el programa y se genere una llamada al Sistema Operativo, quien deberá resolver esta cuestión.

Suponiendo que es tiempo para traer un nuevo proceso a la memoria. El S.O. empieza trayendo uno o varios objetos, incluyendo al que contiene el comienzo del programa. La parte de un proceso que actualmente se encuentra en la Memoria Central, en cualquier momento, se define para ser ***set residente*** del proceso. Cuando el proceso se ejecuta. Las cosas proceden fácilmente con tal de que todas las referencias a memoria sean situaciones que están en el set residente. Usando el segmento o tabla de la página, el procesador siempre puede determinar si esto es así. Si el procesador encuentra una dirección lógica que no está en la memoria central, genera una interrupción que indica una falta de acceso a memoria. El S.O. pone el proceso interrumpido en un estado del bloque y toma el mando. Para proceder después, el S.O. necesitará traer de la memoria central la parte del proceso que contiene la dirección lógica que causó la falta de acceso para la ejecución de este proceso. Esto se conoce como swap-in. Para este propósito, el S.O. emite al disco que leyó una solicitud de E/S. Después de que la demanda de E/S se ha emitido, el sistema operativo puede despachar otro proceso para correr mientras el disco esta realizando la E/S. Una vez que la parte deseada se ha traído a la memoria central, una interrupción de E/S se emite, mientras que el sistema operativo retoma el control del mando, y pone el proceso afectado atrás en un estado de Listo.

A todo esto, existen dos implicaciones:

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

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

Ya que un proceso se ejecuta solamente en la memoria central, esta memoria es referenciada como memoria real. Pero un programador o el usuario percibe un potencial de memoria mucho más grande que la que se asigna en el disco. Esto se conoce como Memoria Virtual que veremos en los próximos puntos. La memoria virtual permite la multiprogramación muy eficaz y releva al usuario de la ajustada restricción de la memoria central.

En resumen:

- Se usaron tablas para la traducción de direcciones
- Se resolvió el problema del mal aprovechamiento del espacio (fragmentación)
- Se acumuló conocimiento y experiencia.
- Se mejoró el hardware (Procesadores, arquitecturas, periféricos) por un lado y por el otro: el S.O. y los lenguajes de programación (Estructuras de datos, algoritmos, compiladores, etc).
- Se incorporó el concepto de interrupción y el swappeo.
- Etc.

5.4. Técnicas de administración con swapping (intercambio) o sea Memoria virtual

Hasta ahora, con las administraciones anteriores, no era posible correr un trabajo hasta que hubiera memoria disponible suficiente para cargar todo su espacio de direccionamiento que necesita ese trabajo para su ejecución.

Esta restricción es importante, ya que los programas, cuyos espacios de direccionamiento son mayores que la existencia de áreas libres no utilizadas no es posible cargarlos ni ejecutarlos.

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

Es una técnica que se utiliza sobre todo en sistemas de tiempo compartido, donde hay más usuarios que memoria para contener todos sus procesos, por lo que es necesario utilizar algún otro soporte para almacenarlos temporariamente. Esta técnica requiere por lo tanto un almacenamiento de resguardo que generalmente se trata de un disco o un medio que permite un acceso directo. Para ser ejecutado, los procesos deben estar cargados en la memoria, pero considerando que no siempre podrán permanecer en ella la totalidad de los procesos que esperan ser ejecutados, se utiliza el intercambio con un disco (por ejemplo), combinando esta técnica con una de selección de procesos a ejecutar. Obviamente como cada intercambio trae aparejado un proceso de E/S, si bien soluciona el problema del tamaño de la memoria, se tendrá retardos en la ejecución de los programas que se harán más lentos.

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

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

Antes de estudiar estos dos métodos, analizaremos el concepto de Swapping, que fuera descubierto por los ingleses en la Universidad de Manchester.

5.4.1. SWAPPING

Swapping es el intercambio de información entre dos niveles de memorias. En este caso Memoria Central y Memoria Secundaria basado en un dispositivo de acceso directo (DASD: Direct Access Storage Device).

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

Se pueden añadir intercambios a cualquier algoritmo. A intervalos determinados por el S.O., generalmente indicados por las políticas de planificación de la CPU, los procesos se copian de la memoria central al almacenamiento auxiliar y más tarde se copian de nuevo a la memoria central. Este esquema permite ejecutar más procesos de los que caben a la vez en memoria.

Normalmente un proceso que sale de un intercambio regresa al mismo espacio de memoria que antes ocupaba. Esta restricción la determina el método de enlace de direcciones. Si el enlace se lleva a cabo durante el ensamble o carga, el proceso no puede moverse a otras localidades. En cambio, si el enlace se realiza durante la ejecución, se puede intercambiar un proceso colocándolo en un espacio de memoria distinto.

Existen distintas variantes de esta técnica de almacenamiento:

- Sistema monoprogramado.
- Multiprogramado con particiones fijas.
- Multiprogramado con particiones variables.
- Sistemas de intercambio de almacenamiento virtual.

Describiremos conceptualmente el primero:

Solamente un usuario utiliza el área de intercambio en la memoria central. Ejecuta hasta solicitar una operación de Entrada/Salida, o se termina su tiempo o finaliza su ejecución. Entonces se lo envía al almacenamiento secundario (swap-out) y se carga otro Programa en su lugar (swap-in).

Los programas deben ser más chicos que el área de intercambio M (restricción).

OBJETIVO: usar la memoria central y la CPU plenamente.

Se pueden añadir intercambios a cualquier algoritmo. A intervalos determinados por el S.O., generalmente indicados por las políticas de planificación de la CPU, los procesos se copian de la memoria central al almacenamiento auxiliar y más tarde se copian de nuevo a la memoria central. Este esquema permite ejecutar más procesos de los que caben a la vez en memoria central.

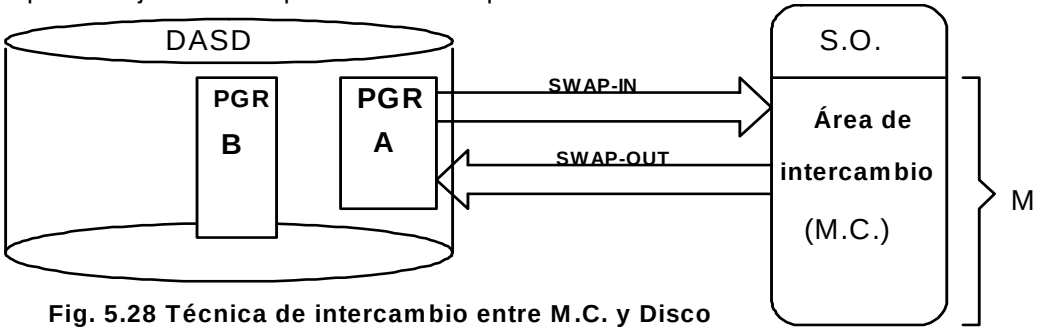


Fig. 5.28 Técnica de intercambio entre M.C. y Disco

Consideremos dos programas A y B, de 20 KBy y 100 KBy respectivamente que se intercambian y supongamos tener un disco con un tiempo de servicio de 10 ms. La tasa de transferencia de datos es de 100 KBy/seg. Calculemos el tiempo de intercambio:

$t_{(swap\text{-}in)}$ = 100 KBy÷100 KBy/seg. + 10 ms = 1,01 seg.
$t_{(swap\text{-}out)}$= 20 KBy÷100 KBy/seg + 10 ms = 0,21 seg.
el tiempo total del intercambio será: $t_{(swap)}$= 1,01 + 0,21 = 1,22 seg.

Comentario: Consideremos un procesador que ejecute 1 MIPS ¿Cuántas instrucciones ejecuta en el lapso de swapping? 1.220.000 instrucciones por lo que se pretende mejorar la performance del sistema. Esto se logra mediante Buffers de entrada y Buffers de salida, por lo que se llega al esquema semejante al de particiones fijas.

Restricciones del Swapping:

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

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

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

5.4.2. PAGINACIÓN POR DEMANDA O BAJO SOLICITUD.

a) Conceptos introductorios

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

La memoria virtual da la impresión de que el sistema tiene más memoria central, haciendo posible que exista una cantidad total mayor de memoria asignada para programas que la memoria central instalada. Esta técnica usualmente funciona debido al principio de localidad.

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

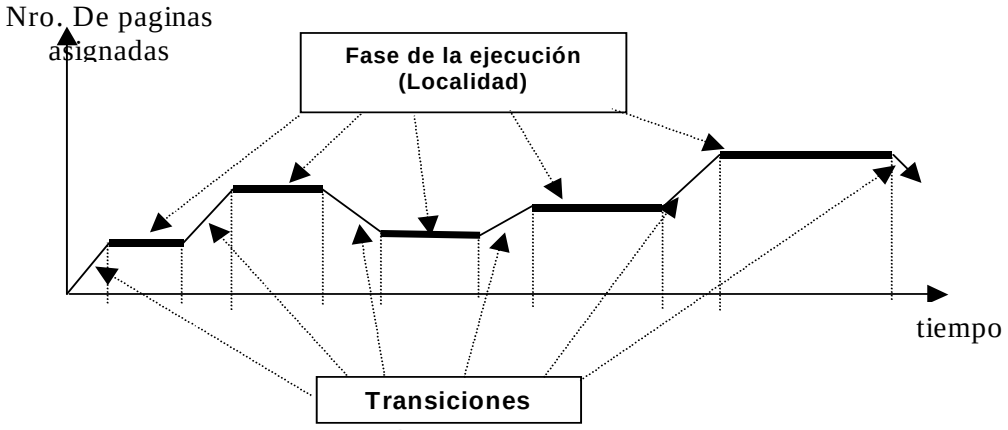


Fig. 5.29. Fases de ejecución y transiciones de un procesador sobre un proceso.

El principio de estados de localidad durante las fases de ejecución, de un dado programa, requiere de un subconjunto pequeño de páginas virtuales durante un tiempo dado. El conjunto de páginas usado durante una fase es el conjunto de localidad de la fase. Nótese que las fases no son uniformes en la longitud (tiempo). Las fases de un programa pueden arbitrariamente definirse como una secuencia de referencias que se hacen a una localidad. Esto se conoce como ***trazas de páginas referenciadas*.** En un programa real se verían estas referencias como se indica en la Figura 5.30

Las direcciones que generan los procesos son ***direcciones virtuales***. Las direcciones de memoria que referencian a la memoria central se denominan "***direcciones reales***". Por más que los procesos hagan referencia a direcciones virtuales, sólo pueden ejecutarse en el almacenamiento primario; por lo tanto las direcciones virtuales deben ser transformadas en direcciones reales mientras el proceso esta en ejecución. Para ello se recurre a un traductor de direcciones llamado **mapper**.

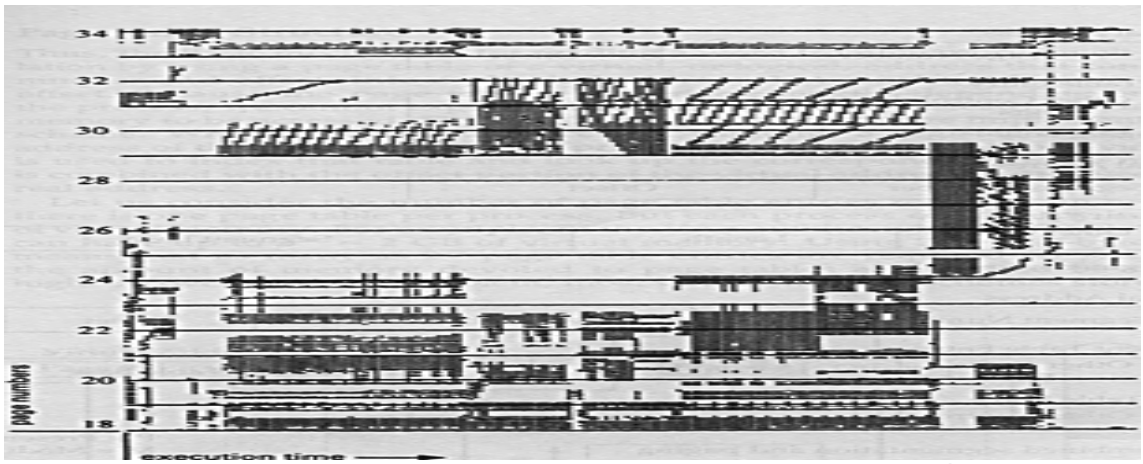


Fig. 5.30 Comportamiento de un programa paginado en ejecución

El mecanismo de traducción de direcciones deberá mantener mapas que ilustren que direcciones del almacenamiento virtual se encuentran en memoria real, y donde se localizan físicamente.

La información se agrupa en **bloques** o **marcos** (**frames**) y el sistema operativo debe estar informado del lugar en que se encuentra almacenada cada página. Cuando los bloques son del mismo tamaño se implementa la paginación. Si los bloques son de tamaño variable se implementa la segmentación.

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

Cuando se planifica inicialmente un trabajo para su ejecución, generalmente solo se carga físicamente sus primeras páginas, y las demás páginas que necesite el trabajo se cargan subsecuentemente bajo solicitud o demanda de la ejecución, por lo que la memoria se asigna y se desasigna durante la ejecución del trabajo.

Cuando un trabajo hace referencia a un área del espacio de direccionamiento que no está en la memoria física, el Hardware de mapeo genera una ***interrupción por falta de página (Page fault).*** El S.O. procesa ésta interrupción, no como una excepción (trap) sino una interrupción que realiza el servicio de cargar las páginas requeridas desde el almacenamiento secundario (swap-in) y actualiza en forma adecuada las entradas de la tabla de páginas, luego reinicia la ejecución de la instrucción interrumpida. Las páginas solicitadas están cargadas en un dispositivo de almacenamiento secundario donde se ha reservado una copia de todo el espacio de direccionamiento del trabajo.

Una vez que la memoria se ha llenado con páginas y ante la demanda de un espacio adicional, solo es posible cargar una serie de páginas nuevas desde el disco si previamente se elimina primero algunas de las que ya estén residentes en la memoria (esto se plantea en la Fig. 5.31). Las páginas reemplazadas (víctimas) se copian al dispositivo de almacenamiento secundario antes de cargar la nueva si es que sufrieron alguna modificación durante su procesamiento. Esta técnica se llama ***sustitución, intercambio o trueque de página***.

Los algoritmos que deciden que página remover han sido el centro de extensas investigaciones y apuntan a evitar el fenómeno de mover excesivamente las páginas entre la memoria y los dispositivos de almacenamiento secundario *(conocido como **azotado, hiperpaginación o thrashing*** que estudiaremos mas tarde en éste módulo*)* ya que genera un gran overhead. Esto se conoce como degradación de la multiprogramación (***Degree of Multiprogramming*****)** que es el número de procesos activos que hay en el sistema compitiendo por el uso del procesador. De aquí surgen dos nuevos elementos a tener en cuenta:

- **CPU Utilization:** El porcentaje de tiempo que el CPU está ocupado y no está esperando para periféricos para completar de procesar un trabajo.
- **Load:** El número de procesos activos que hay en el sistema.

Para ver el desempeño del sistema se definen estos conceptos:

- **Memory Queue:** Las transacciones tienen lugar en la cola de memoria hasta que los recursos en el sistema sean liberados para hacer la transacción.

- **Multiprogramming Level (MPL):** El número de procesos activos que hay en el sistema.
- **Response Time:** La cantidad de tiempo que un procesos esta con un servidor, incluyendo el tiempo de servicio.
- **Server:** También se refiere a un dispositivo. Un componente del sistema que mejora el trabajo para las transacciones que realizan los procesos
- **Service Time:** La cantidad de tiempo que un servidor se toma para completar el trabajo en una tarea y devolver el servicio.
- **Throughput:** El número de transacciones o tareas completas por el sistema o por el servidor por segundo.
- **Visit Ratio:** El número de veces que una transacción o tarea es solicitado por un servidor antes de salir del sistema.

Estos conceptos lo ampliaremos mas adelante pero sirven para diseñar el administrador quien deberá considerar estos parámetros para una eficiente ejecución del sistema. Por ejemplo mantener un adecuado nivel de multiprogramación para procurar minimizar las fallas de páginas y por consiguiente no degradar el sistema. Se suele recurrir al siguiente modelo de administrador:

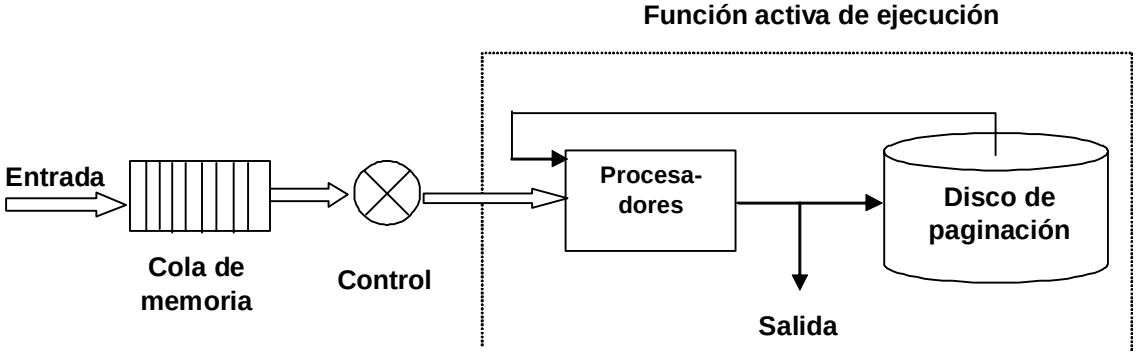


Fig. 5.31. Modelo de administrador de paginación por demanda

La función de control regula la carga del sistema de forma que el desempeño esté en el límite acotado por el throughput, el tiempo de respuesta, etc., que se haya fijado como estrategia de administración.

Para ayudar al S.O. a tomar las decisiones de sustitución, generalmente el Hardware mantiene cierta información acerca del uso de las páginas a través del mapeo de direcciones mediante información adicional que se incorpora a las tablas de mapeo de páginas. Para cumplir con esta tarea se necesita tres cosas:

- Un **bit de estado** en la tabla del mapa de páginas que indique si la página está en memoria central o en el almacenamiento secundario.
- Una acción de interrupción que transfiera el control al S.O. si el trabajo trata de acceder a una página que no está en la memoria central.
- Un registro del uso de las páginas individuales para que el S.O. pueda determinar cuales eliminar cuando sea necesario.

La administración de memoria paginada por demanda puede lograr lógicamente una utilización de memoria central superior al cien por cien, o sea, que la suma de todos los espacios de direccionamiento de los trabajos multiprogramados puede superar el tamaño de la memoria física.

Resumiendo:
Memoria virtual: es una técnica que permite simular una memoria central mayor de la que realmente hay disponible en el Sistema, usando un dispositivo de memoria secundaria (generalmente un disco rápido) conjuntamente con un mecanismo de extensión del direccionamiento y se puede ver como la separación de la memoria lógica del usuario de la memoria física.

b) Las funciones de administración de memoria paginada bajo demanda:

1) Llevar control del estado: se logra mediante tres tablas:
   a) Tablas del mapa de páginas (MPT o PT): una por espacio de direcciones o sea por trabajo.
   b) Tablas de bloques o frames de memoria (MFT): una en el sistema.
   c) Tablas de mapas de archivos: una por espacio de dirección del trabajo en el disco, generalmente está asociado con la MPT.

2) La política de quién y cuándo obtiene la memoria: Es determinado parcialmente por el planificador de los trabajos y en forma dinámica por las interrupciones generadas por las fallas de página bajo demanda.

3) Asignación: cuando se requiere asignar un bloque, es necesario encontrar uno disponible y modificar su estado que pasará de libre a ocupado en la Tabla de frames MFT.

4) Desasignación: si no es posible encontrar un bloque disponible para la asignación, se debe desasignar y reasignar uno de los bloques asignados. Al terminar un trabajo se desasignan todos los bloques que tenía asignado dicho trabajo durante su ejecución.

c) Ventajas:

1. Los Usuarios pueden disponer de grandes cantidades de memoria central con lo que se logra una mayor multiprogramación.
2. La Programación se ve simplificada, el programador no necesita comprimir el programa para que quepa en memoria central.
3. La memoria central es utilizada en un factor superior al 100%.
4. Programas largos que no caben en la memoria central disponible pueden ejecutarse.

d) Desventajas:

1. Baja performance del sistema debido al uso de E/S para suplir la escasez de Memoria Central.

e) Características:

En este sistema los procesos residen en memoria secundaria (habitualmente un disco). Cuando se quiere ejecutar un proceso se introduce en la memoria las páginas necesarias. Esto lo hace el ***páginador (Pager)*** quien trata con las páginas individuales de un proceso, evitando colocar en la memoria páginas que no se utilizarán, reduciendo el tiempo de intercambio y la cantidad de memoria física necesaria.

En la tabla de páginas se agrega un bit adicional válido-inválido que indica la presencia, o no, de esa página en Memoria Central. Si este bit está asignado como “válido”, indica que la página asignada se encuentra instalada en memoria, si es “inválido” la página está en disco.

Mientras el proceso se ejecuta y accede a páginas ***residentes en memoria***, la ejecución prosigue normalmente.

Cuando se hace referencia a una página que no está en memoria, el Hardware produce una interrupción por la falta de página en memoria central (**page fault o falla de página**) y el S.O. carga la página solicitada nueva en Memoria. Es importante considerar que un error de dirección no válida es consecuencia de intentar utilizar una dirección de memoria ilegal; en tal caso el proceso deberá terminar como un trap. Sin embargo, en esta situación la interrupción es el resultado de la falla del S.O. al no transferir a memoria una parte válida del proceso, tratando de minimizar el tiempo adicional de transferencia de disco y los requisitos de memoria.

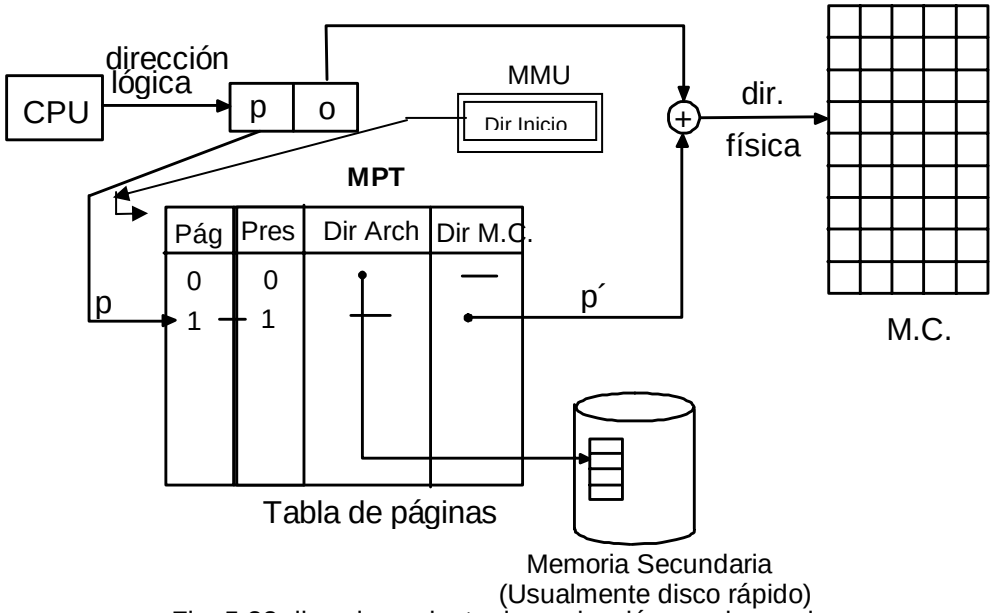


Fig. 5.32 direccionamiento de paginación por demanda

Una observación muy importante es la siguiente: El S.O. tratará de eliminar en una sola operación la mayor cantidad de páginas que no se usan para hacer lugar en la memoria central. La razón es bajar el overhead de esta operación si se ejecutara cada vez que se necesite el espacio de una sola página. De igual forma actúa cuando trae o graba páginas en el disco. Genera una sola operación de E/S y “swapea” un conjunto de páginas.

f) Requisitos de hardware para la administración:

1. En la MPT se le incorpora un bit de presencia.
2. El Hardware debe producir una interrupción cuando una página referenciada no está en la memoria central.
   Esta interrupción es tratada por el S.O. y el sistema de interrupciones del procesador (First Level Interrupt Handler) que procede a buscar en el disco un conjunto de páginas, cargarlas en memoria central y actualizar las tablas. En caso de que no hubiera lugar en memoria central, el S.O. debe decidir cuáles páginas seleccionará como "víctimas" (a reemplazar) mediante un algoritmo de reemplazo de páginas.
3. Descripción del uso de las páginas para asistir al S.O. en la decisión de qué página sustituir (Ver Algoritmos de Sustitución).
4. Las instrucciones del hardware deben poder recomenzar si ocurre una interrupción por page fault durante su ejecución.
5. Brindar una memoria secundaria que contenga las páginas que no se encuentran en memoria central (Almacenamiento auxiliar).

g) Requisitos de software:

1. Debe existir una interacción con los administradores de Entrada/Salida y del sistema de archivos.
2. Además del bit de presencia, se necesita conocer la dirección de archivo que contiene la página y su desplazamiento dentro del archivo para saber en qué lugar del disco se encuentra la página faltante.

5.4.3. Políticas de Administración de Memoria Virtual

Para administrar eficientemente la memoria, hay que considerar al hardware disponible. ¿Apoya éste el uso de páginas, segmentos o ambos? Esto permite decidir si el sistema operativo puede manejar memoria virtual. Si esto es posible, entonces hay que estudiar que políticas o algoritmos se implantarán para conseguir administrar la memoria de forma eficiente y a la vez transparente al usuario.

- **Políticas de "fetch" o búsqueda**: Estas políticas se implantan en un sistema operativo para decidir cuándo debe traerse una página a memoria. "**Demand paging**" es la política que determina que se debe cargar a memoria una página cuando ésta se necesita, es decir, cuando se genera una dirección en esa página y hay que cargarla para que continúe la ejecución del proceso. Con "anticipatory paging" o "**prepaging**" se trata de traer una página a memoria antes de que ésta se necesite (para evitar una interrupción en la ejecución del proceso). Se ha determinado que el "prepaging" no hace al sistema significativamente más eficiente, pues su posible ventaja de aligerar el sistema al minimizar interrupciones se neutraliza por la complejidad de los algoritmos usados para elegir la página a traer, y por el trabajo adicional que conlleva cargar páginas que no se van a usar.
- **Políticas de "placement" o colocación**: Estas políticas se implantan en un sistema para decidir en qué dirección de memoria se cargará la página a traerse a memoria. En el caso de páginas es simple pues todas son del mismo tamaño. En el caso de los segmentos se implantan algunos posibles algoritmos, ya vistos con anterioridad: "best fit", "worst fit", "first fit", "next fit".
- **Políticas de "replacement" o reemplazo**: Estas políticas se utilizan para decidir, en caso de que haya que traer una página a memoria primaria y ésta esté llena, qué página debe sacarse de memoria para hacer espacio a la nueva que llega. Para tomar la decisión más acertada en este aspecto, debe tenerse en consideración cuántas páginas de cada proceso pueden cargarse a la vez a memoria (mientras más páginas haya se generan menos interrupciones pero se desperdicia más espacio con páginas que tal vez no se estén usando), o sea, cuál es el "working set" del proceso; también debe considerarse si la posible página a sacarse debe pertenecer al proceso cuya página se cargará o no.

Secuencia de una falla de página (page fault):

Supongamos mientras un proceso está ejecutando, ocurre un fallo de página. Los eventos se representan en la Figura 5.34. El hardware interrumpe al S.O., el cual verifica sus tablas internas para ver si se trata de un fallo de página ❶ o de un acceso ilegal a memoria. El S.O.❷ determina donde reside la página requerida en el disco, pero luego no encuentra frames libres sobre la lista de frames libres❸, es decir, toda la memoria está en uso.

El reemplazo de página❹ se fundamenta en lo siguiente: si ningún frame está libre, encontrar al menos uno que no está siendo usado actualmente y lo libera; para ello escribe su contenido en el disco si fue modificada, sino la sobreescribe y cambia la tabla de páginas para indicar que la página no está

más en memoria❺. El frame liberado puede ser usado ahora para alojar la página por la cual el proceso falló❻.

Ocurre una interrupción cuando la dirección generada por el procesador localiza la correspondiente entrada a la tabla MPT y encuentra el bit de presencia desactivado, entonces se genera la siguiente secuencia:

1. Interrupción por falta de página en memoria central(page fault)
2. Si la dirección no es legal se produce un trap de protección.
3. Busca espacio libre en memoria central (si no hay, hace lugar eligiendo víctimas según el algoritmo de sustitución y si hay espacio libre, usarlo, además si la víctima sufrió una modificación durante su procesamiento, entonces escribir la página víctima sobre el disco, actualizar las tablas de páginas y frames).
4. Lee la página requerida del disco y la carga en memoria central dentro del frame liberado.
5. Corrige y actualiza las tablas MPT (Memory Page Table) y MFT (Memory FrameTable) con la nueva página traída y el lugar ocupado
6. Recomienza la instrucción interrumpida del proceso de usuario.

Esta secuencia se esquematiza en las figuras 5.35 y 5.36.

Al incrementar el grado de multiprogramación se produce una ***sobreasignación*** de memoria.

Una importante observación: El tiempo medio de acceso (tma) a memoria central se ve incrementado por la presencia de page faults. Consideremos los siguientes tiempos:

- tma: Tiempo medio de acceso. Usualmente 10 ns ≤ tma ≤ 150 ns

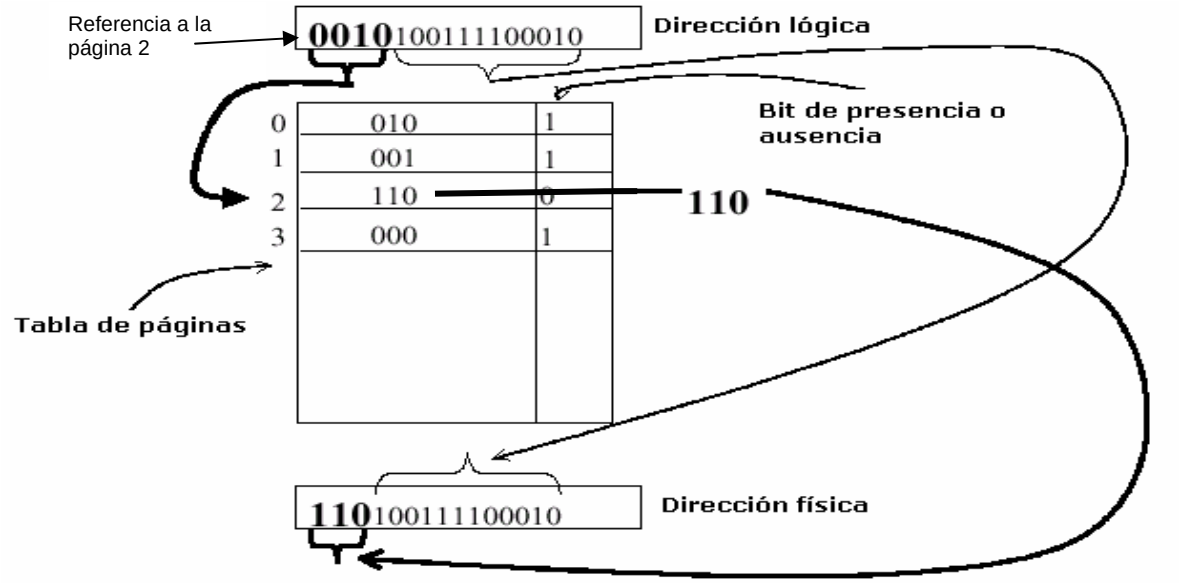


Fig. 5.33 construcción de un page fault

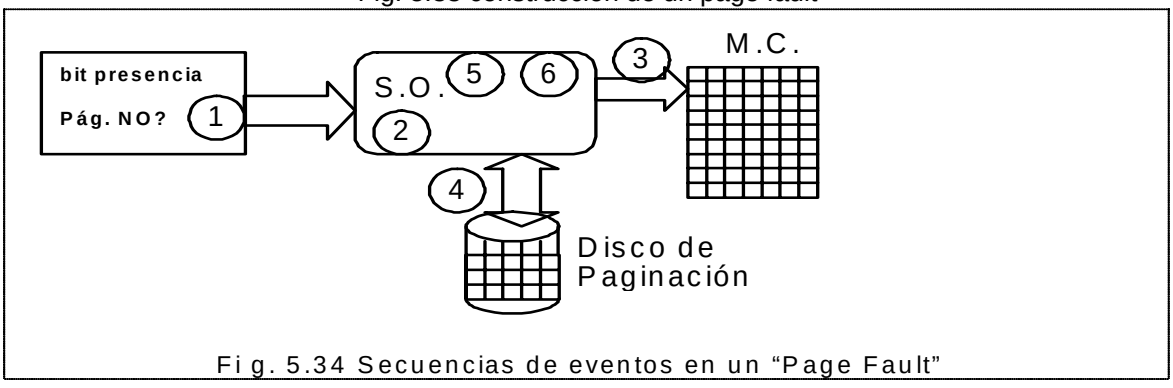


Fig. 5.34 Secuencias de eventos en un "Page Fault"

- tpf: Tiempo de page fault. Tiempo necesario para buscar una página en disco, cargarla en memoria central y actualizar la MPT.
- tpf = t. latencia + t. seek + t. transferencia ≈ 10 ms + 15 ms + 1 ms = 26 ms para un disco de 6000 RPM
- p_f: Probabilidad de ocurrencia de un page fault. ($0 \le p_f \le 1$)

$$\text{tma real} = (1 - p_f) * \text{tma} + p_f * \text{tpf}$$

Esta es la razón de que se reemplazarán la mayor cantidad de páginas de una sola vez entre el disco y la memoria central para disminuir el tiempo de intercambio que es overhead puro.
Si el S.O. determina en qué lugar del disco está la página (por page fault) y descubre que ya no hay frames libres, es decir toda la memoria está ocupada, tiene varias opciones:

- Abortar el proceso de usuario: esto perjudicaría al objetivo de la paginación por demanda: no mejoraría la utilización y productividad del sistema de computación y no sería transparente al usuario.
- Descartar un proceso liberando todos sus frames y reduciendo el nivel de multiprogramación.
- Sustituir páginas en memoria.

b) Sustitución de páginas:

El reemplazo de página es básico para la paginación por demanda. Esta forma de administrar la memoria central completa la separación entre las memorias lógica y física. Con este mecanismo, una memoria virtual muy amplia puede ser provista sobre una memora física más pequeña. Sin embargo, se deben resolver dos problemas para implementar la paginación por demanda: se debe desarrollar un *algoritmo de reemplazo de página* y uno de *asignación de frame*. Si existen múltiples procesos en memoria se debe decidir cuantos frames se asignan a cada proceso. Así, cuando se requiere el reemplazo de páginas se busca seleccionar solo aquellas que están en condiciones de ser reemplazados. El diseño de estos algoritmos debe hacerse cuidadosamente, ya que las operaciones sobre disco son costosas en tiempo.

Las características y condiciones más importantes que se debe tener en cuenta para el reemplazo de páginas son:

1. Ocurre solo cuando no hay frames libres, o sea, espacio disponible para asignar en memoria central cuando se carga la nueva página demandada.
2. El S.O. elige una serie de páginas a reemplazar, graba en disco aquellas que fueron alteradas y trae las nuevas.

Esta actividad implica que el tma estará duplicado, por lo tanto se usa un **bit de modificación** asociado con cada página para que el S.O. sepa si éste ha sido modificado o no y entonces solo graba aquellas páginas que tiene prendido este bit.

Este bit adicional de modificación en la MPT es alterado por el hardware que lo pone en 1 con cada acceso a memoria central cuando se produce una alteración de los contenidos (de la página) originalmente cargados. Cuando se selecciona una página para ser reemplazada, se examina este bit de modificación. Si el bit está activo (1) indica que la página ha sido modificada y se debe "salvarla", o sea, escribirla completamente en disco. En caso contrario, la página no ha sido modificada por lo tanto también está en el disco y se la puede sobreescribir en memoria central, de esta forma se evita el proceso de escritura en el disco, reduciéndose a la mitad el tiempo de E/S si la página no ha sido modificada.

Entonces se presentan dos grandes problemas para usar paginación por demanda:

1- ¿Cómo asignar frames?
2- ¿Cómo sustituir páginas cuando hay page fault y la memoria está llena?

Antes de estudiar estos dos problemas veremos como la rutina de servicio de falla de página se modifica para incluir el reemplazo de página según la siguiente secuencia:

1- Encontrar la posición de la página requerida sobre el disco.
2- Encontrar un frame libre:
a si hay un frame libre, usarlo
b de lo contrario, usar un algoritmo de reemplazo de páginas para seleccionar las víctimas.
c escribir las páginas víctimas que fueran alteradas sobre el disco, actualizar las tablas de páginas y frames.
3- Leer las páginas requeridas dentro de los frames liberados, actualizar las tablas de páginas y frames.
4- Comenzar el proceso de usuario.

Las técnicas orientadas a la programación basada en objetos, permiten el uso de muchos programas pequeños y módulos de los datos muy referenciados en un período relativamente corto de tiempo. Por otro lado las aplicaciones de Multithreaded producen cambios abruptos en el flujo de instrucciones y en las referencias de memoria esparcidas. Esto requiere un fuerte apoyo del Hardware mediante el uso de TLB y Caché como se explico en paginación pura. Pero aquí corresponde ampliar un poco estos conceptos debido al proceso de direccionar la instrucción lógicamente localizarla físicamente.

Para un tamaño dado de (TLB), como el tamaño de memoria de procesos crece y como la ubicación decrece, el ***hit ratio*** de acceso TLB declina.

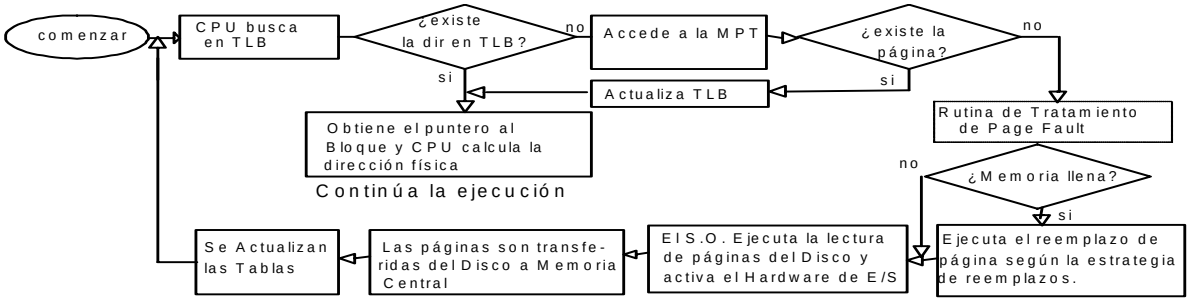


Fig. 5.35 Algoritmo en Paginación por demanda con TLB.

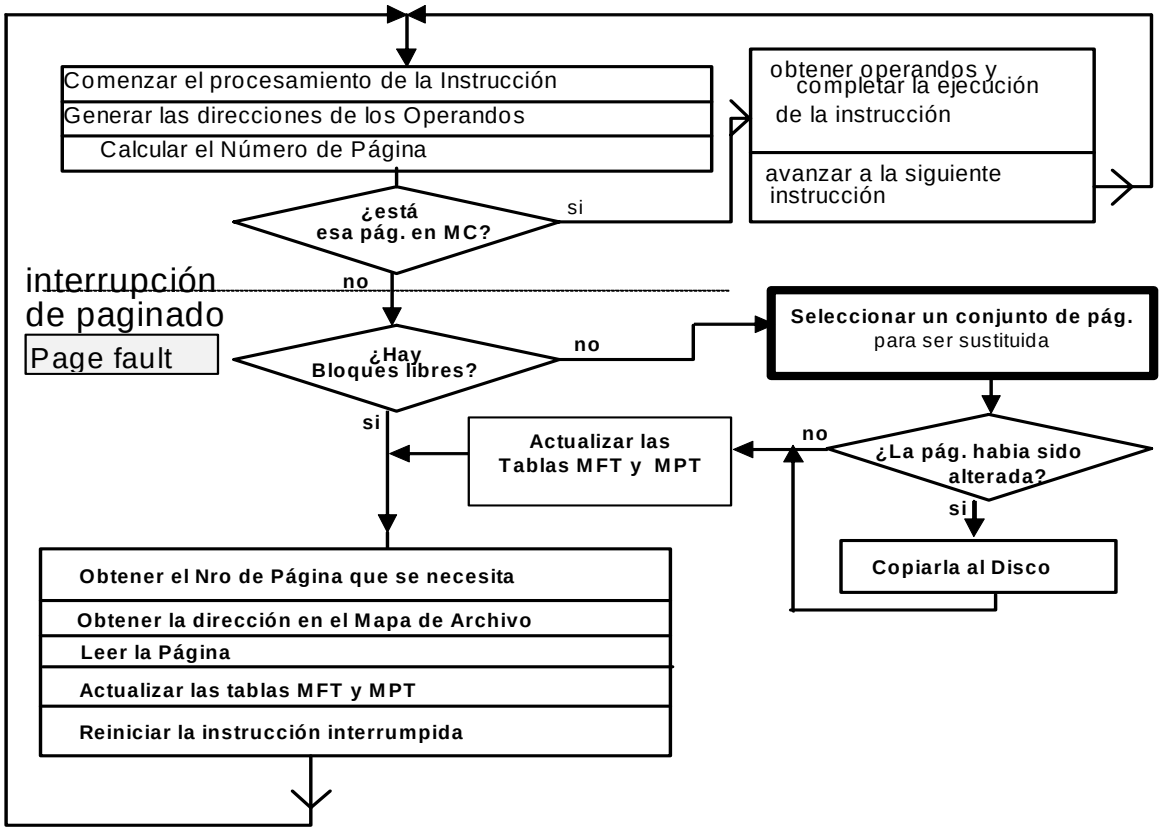


Fig. 5.36 Algoritmo general de paginación por demanda

Una manera de mejorar la performance del TLB es usar un TLB grande con más entradas. Sin embargo, el tamaño del TLB interactúa recíprocamente con otros aspectos del hardware, tales como la caché de la memoria central y el número de accesos a memoria por ciclo de instrucción. El tamaño del TLB es improbable ya que crece tan rápidamente como la memoria central.

Una alternativa es usar los tamaños de la página más grandes para que cada entrada de la tabla de página en el TLB se refiera a un frame (bloque) más grande de memoria. Pero el uso de tamaños de la página grandes puede llevar a la degradación de la performance y a aumentar la fragmentación interna.

De acuerdo con, varios diseñadores han investigado el uso de tamaños de páginas y varias arquitecturas de microprocesadores apoyan tamaños de páginas múltiples, como ser el R4000, Alfa, SuperSPARC, y Pentium, etc. Pero por lo general es de 4Kiby.

Los tamaños de páginas múltiples proporcionan la flexibilidad necesaria para usar un TLB eficientemente.

Esta secuencia la esquematizamos en la figura 5.35, en que diferenciamos la existencia de M.M.U. con y sin TLB.

El algoritmo general de paginación por demanda se presenta en la Figura 5.36 en que la primera pregunta si se encuentra en memoria incorpora el de la figura 5.35.

c) Tipos de Sustitución de páginas:

Objetivo: presentar la menor cantidad de page fault posible.

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

Sustitución global (Asignación dinámica de frames)

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

Sustitución local (Asignación estática de frames)

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

Generalmente es mejor el reemplazo global (es más productivo)

d) Asignación de Bloques (frames)

Un hecho notable en los sistemas que manejan paginación es que cuando el proceso comienza a ejecutarse ocurren un gran número de fallos de página, porque es cuando está referenciando muchas direcciones nuevas por vez primera, después el sistema se estabiliza, conforme el número de marcos asignados se acerca al tamaño del conjunto de trabajo.

Observación: de acuerdo a cada arquitectura, cada proceso requerirá un número mínimo de páginas (frames) disponibles. Si no el programa no puede ejecutarse eficientemente. Por lo que es importante asignar correctamente el número de frames para así obtener una organización más eficiente.

⇒ El Número de frames mínimo y necesario para la ejecución de un trabajo tiene las siguientes restricciones:

- No podemos asignar más del total de frames libres (excepto con páginas compartidas).
- Existe un número de frames mínimo que puede asignarse.
- Si se reduce el número de frames asignados a cada proceso, aumenta los page fault, haciendo más lenta la ejecución de los procesos.

⇒ El número mínimo a asignarse está definido por:

- La arquitectura del conjunto de instrucciones. Se debe contar con frames suficientes para todas las páginas a las cuales pueda hacer referencia una instrucción.
- La arquitectura del computador: referencias indirectas, niveles de indirección, etc.

⇒ El número mínimo de frames por proceso está definido por la arquitectura, mientras que el número máximo está definido por la cantidad de memoria física disponible.

Para la **asignación** se utilizan dos técnicas principales:

1. **Asignación pareja o equitativa**: si hay m frames y n procesos. Se le da a cada proceso **m/n** frames.
2. **Asignación proporcional**: se asignan frames de acuerdo al tamaño de cada proceso.

Si cada proceso mide t_i bytes y los frames son de t_f bytes, el número de frames asignado al proceso será:

$$f_i = t_i / t_f$$

De esta manera se reconoce que los diversos procesos necesitarán cantidades distintas de memoria.

En ambas técnicas las asignaciones pueden variar de acuerdo con el nivel de multiprogramación. Al aumentar el nivel de multiprogramación, cada proceso perderá algunos frames para proporcionar al nuevo proceso la memoria necesaria. Si disminuye el nivel de multiprogramación, los asignados al proceso que sale puede distribuirse entre los restantes.

Se trata de igual forma a un proceso de alta o baja prioridad. Pero se quiere dar más memoria al proceso de alta prioridad para acelerar su ejecución. Para esto, se puede:

- Usar un esquema de asignación proporcional donde la tasa de frames no depende de tamaños relativos de los procesos, sino de sus prioridades o de una combinación de ambas.
- Permitir que un proceso de alta prioridad seleccione para su reemplazo los frames de un proceso de baja prioridad. Un proceso puede seleccionar un reemplazo entre sus propios frames o entre los de cualquier proceso de menor prioridad.

	Reemplazo local	***Reemplazo global***
Asignación fija	El número de frames asignados al proceso es fijo. La página a ser reemplazada es elegida entre los frames asignados para ese proceso.	No es posible.
Asignación variable	El número de frames asignados a un proceso puede ser cambiado de tanto en tanto, para mantener el set de trabajo del proceso. La página a ser reemplazada es elegida entre los frames asignados a ese proceso.	La página a ser reemplazada es elegida entre los frames disponibles en Memoria Central; esto causa que varíe el tamaño del set residente de procesos.

Tabla 5.3 Resumen de las políticas de asignación y reemplazos

e) Prepaginación

Consiste en traer páginas por adelantado para evitar un gran número de page faults.

Para decidir si conviene la prepaginación se debe comparar el costo de traer inicialmente n páginas versus el costo de producir page faults por cada página que no se trae (y considerar que exista un lugar donde ponerlas).

Suponiendo que las páginas están prepaginadas y una fracción **x** de estas **n** páginas realmente se usan **(0 ≥ x ≥ 1)**. La pregunta es si el costo de las **xs** fallas de páginas que se evitan es mayor o menor al costo de la prepaginación de **(1 - x) s** páginas innecesarias. Si **x** está cerca del cero, no conviene la prepaginación.

f) Hiperpaginación o Thrashing

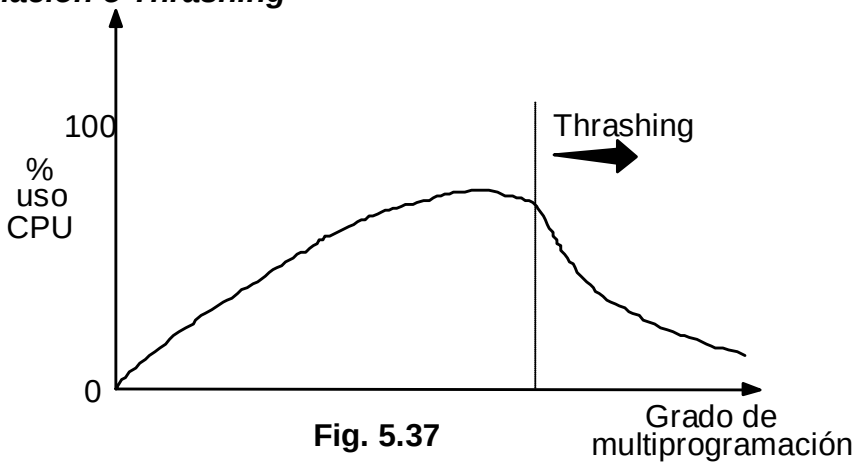


Fig. 5.37

Se dice que un proceso entra en thrashing (azotado, apaleamiento o sacudones) cuando la cantidad de page faults es tan elevada que gasta más tiempo paginando que ejecutando.

Una posible causa de thrashing:

Supongamos que se utiliza sustitución global de páginas y que cuando el porcentaje de uso de CPU cae (aumenta la cola de espera sobre el dispositivo del paginador), el S.O. aumenta el grado de multiprogramación (porque la cola de dispositivo se vacía). ¿Que sucede?

- la tasa de page faults aumenta considerablemente;
- se incrementa el tiempo de acceso efectivo a memoria;
- la utilización del procesador decae;
- no se realiza ningún trabajo ya que los procesos se dedican a paginar.

Soluciones:

1. Reducir el nivel de multiprogramación mediante un ***algoritmo de reemplazo local*** (o por prioridades). De esta manera, si un proceso entra en thrashing, no puede robar frames de otro

proceso y provocar que éste también caiga en thrashing. Sin embargo, si los procesos están en thrashing, la mayor parte del tiempo puede encontrarse en la cola del dispositivo de paginación y el tiempo de acceso efectivo aumentará para todos los procesos.

2. Deben tratar de proveer al sistema la cantidad de frames suficientes para que no ocurra thrashing. Esto se hace sobre la base de la tasa de thrashing aceptable y las decisiones que toma el S.O.

g) Modelo de Conjuntos de trabajo (Working Set - WS) y conjuntos residentes

Recordemos que Localidad es un conjunto de páginas usadas por un proceso en un dado momento.

Un programa generalmente está compuesto por varias localidades distintas. Las localidades están definidas por la estructura del programa y sus estructuras de datos. Si se asigna menos frames que el tamaño de la localidad el proceso entrará en thrashing, ya que no mantiene en memoria todas las páginas que se están usando.

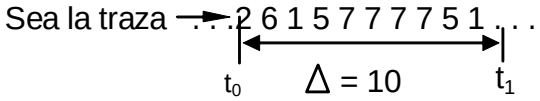
Definimos **conjunto residente** en el momento t_1 como el conjunto de páginas cargadas en frames en la memoria central.

El modelo establece que un programa al ejecutar cambia de localidad en localidad por lo tanto para que no ocurra thrashing hay que mantener todos los frames de la localidad en memoria central hasta que el proceso cambie de localidad.

La solución consistirá en tratar de aproximar la localidad de un proceso.

Definimos **Working Set Window** en el momento t_1 como el conjunto de páginas usadas en las últimas Δ referencias

Ejemplo:



$$WSW(t_1, \Delta) = \{1, 2, 5, 6, 7\}$$

Fig. 5.38 Ventana de 10 frames

- Si Δ es muy chico, el WSW no cubre el Working Set real y por lo tanto el proceso tendrá menos páginas de las que necesita.
- Si Δ es muy grande, el WSW cubre más de un WS real y por lo tanto el proceso tendrá asignados frames que no utiliza.
- Si $\Delta = \infty$, el WSW cubre todo el espacio de memoria del proceso.
- Cada proceso i requiere un WSW de tamaño TWS_i por lo que la demanda global de frames por parte de los procesos, es:

$$D = \sum_i TWS_i$$

Por lo tanto si la demanda D es superior al tamaño de la memoria central, es probable que en el sistema ocurra thrashing.

Se puede aproximar a este modelo mediante interrupciones a intervalos fijos de un cronómetro y un bit de referencia.

Ventajas:

1. El criterio para evitar thrashing es el siguiente: Si el WS de un nuevo proceso cabe en memoria central entonces lo ejecutamos. De esta forma estamos incrementando el grado de multiprogramación sin incurrir en thrashing.
2. Evita el thrashing manteniendo a la vez el nivel de multiprogramación lo más alto posible.
3. Optimiza la utilización del procesador.

Desventajas:

El problema con esta técnica es que TWS_i es dinámico y es muy costoso en cuanto a recursos, estimar su valor. Por lo tanto, éste modelo tiene más valor teórico que práctico y se lo utiliza como patrón de comparación con otras técnicas.

5.4.4. Algoritmos de reemplazo de páginas

Dado que existen diferentes algoritmos de reemplazo de página, la selección de uno de ellos implica una evaluación que consiste en correrlo sobre una cadena particular de referencias de memoria y

computando el número de fallas de página. Para determinar dicho número, necesitamos conocer el número de frames disponibles. Obviamente, al incrementar el número de frames disponibles, se decrementa el número de fallas de página.

Entonces para determinar cual de ellos es el mejor algoritmo se consideran:

1. El punto de comparación es exclusivamente las listas de referencias a memoria que realiza el programa a medida de que se ejecuta.
2. El algoritmo se evalúa ejecutándolo, dando como entrada una lista particular de referencias a páginas producidas por la ejecución del programa usuario. Esto se registra y suele llamarse ***trazas de referencias de páginas*.**
3. Se necesitan tres parámetros
   a) Traza de páginas P.
   b) Un determinado tamaño de memoria M.
   c) Un Algoritmo de reemplazo R.

Para determinar la **Performance de un algoritmo de sustitución** usaremos la siguiente notación:

- S: Número de page hits o aciertos.
- F: Número de page faults o fallas.
- PP: Número total de referencias.

PP = S + F
h = S / PP (Hit ratio o success frecuency function)
f = F / PP (Page fault ratio)

Con estos datos y sus fórmulas podemos comparar los distintos algoritmos de reemplazo de páginas.

a) - Algoritmo de reemplazo óptimo (OPT o MIN)

Objetivo: Consiste en organizar los reemplazos de manera tal que se reemplace la página que no va a ser usada por el máximo período de tiempo.

Funcionamiento:

- En cada instante i, se calcula el tiempo Tj que falta para que se realice una referencia a la página Pj.
- Se sustituye la página para la cual Tj es máximo.

Ventajas:

1. Tiene la mejor tasa de fallas de páginas de todos los algoritmos.

Desventajas:

1. - Costosa de implementar, sólo se la usa para comparar con otros algoritmos.
2. - Además, necesita “conocimiento futuro”, lo cual es difícil de adquirir.

Ejemplo:

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	Serie de referencias
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7	
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0	MARCOS
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1	
			F		F		F			F			F				F			FALLOS DE PÁGINAS

Fig. 5.39 Algoritmo de reemplazo de Páginas OPT

Se observa en la Fig. 5.39 que se producen 6 fallas de páginas (F) que se podrían haber evitado entregando al proceso 6 marcos de páginas por anticipado en lugar de 3. En ese caso las fallas de páginas serían cero.

Observación: se consideran que los 3 frames iniciales fueron asignados y se cargaron las páginas sin producirse page fault.

b). Algoritmo FIFO (first-in, first-out)

Objetivo: Elegir para sustituir la página que ha permanecido más tiempo en memoria.

Funcionamiento:

- Se elige la más antigua.
- Para saber cual es la más vieja:

1. Asociar hora de carga a cada página o una marca de tiempo (timer), que es más chico y más barato.
2. Llevar una cola FIFO de las páginas que hay en memoria. Se reemplaza la página al inicio de la cola, y cuando se introduce en memoria una página, se inserta al final de la cola.

Ventajas:

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

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

Desventajas:

1. Presenta la llamada ***Anomalía de Belady***: bajo ciertas trazas, el page fault ratio crece cuando se agregan más frames a la memoria central, o dicho de otro modo, dada una traza, cuánto más frames se asignan al proceso, más fallos de páginas produce.
   Si se considera la siguiente serie de referencias: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, y 5 que se grafican en la Fig. 5.41.
   Observando el gráfico de la figura 5.40 se ilustra como al aumentar el número de frames a 4 el número de page fault aumenta con respecto a 3 frames.
2. La página reemplazada puede contener una variable cuyo valor inicial se asignó hace tiempo pero que se utiliza constantemente.

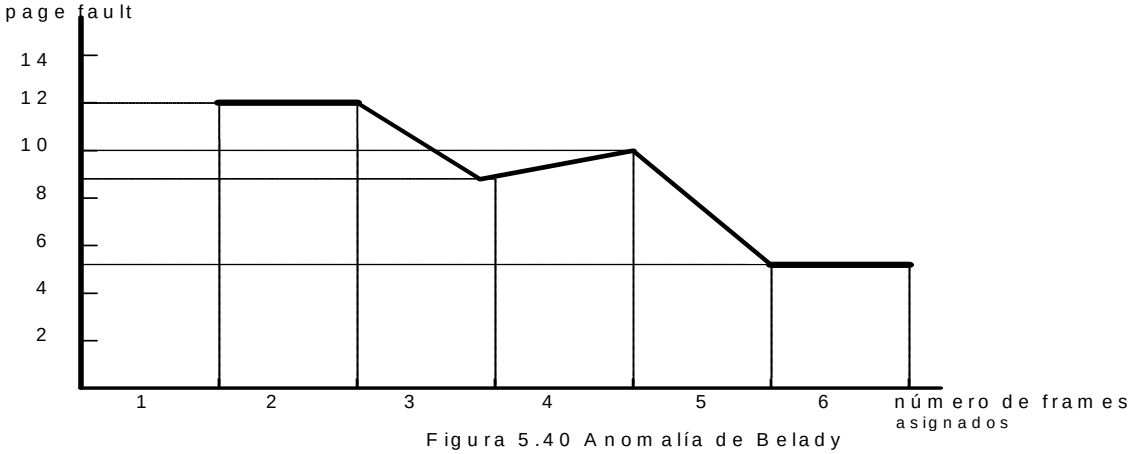


Figura 5.40 Anomalía de Belady

Ejemplo: (mantenemos las mismas referencias que en la figura 5.39 para FIFO):

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	Serie de referencias
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7	
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0	MARCOS
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1	
			F		F	F	F	F	F	F			F	F			F	F	F	FALLOS DE PÁGINAS

Fig. 5.41 Algoritmo de reemplazo de Páginas FIFO

Se producirán 12 fallas de páginas con los 3 frames asignados.

c). LRU Least Recently Used

Objetivo: Reemplazar la página que no se ha utilizado durante el mayor período de tiempo.

Funcionamiento:

- Se basa en el principio de que si una página ha sido usada, será requerida nuevamente muy pronto, entonces las sustitutas a ser reemplazadas son las que menos se han usado recientemente.
- Asocia a cada página el instante en que se usó por última vez.
- Se necesita llevar un registro del uso de las páginas y esto lo debe hacer el Hardware.

Ventajas:

1. No sufre la anomalía de Belady. Pertenece a la clase de algoritmos llamados **Algoritmos de pila**.
2. Se puede demostrar que el conjunto de páginas en memoria para n frames es igual al conjunto de páginas para n + 1 frames. Ya que con n frames tendríamos en memoria central las n páginas más usadas recientemente y con n + 1 frames pasaría lo mismo. Esto no era cierto con FIFO.
3. Se necesita llevar un registro del uso de las páginas y esto lo debe hacer el Hardware a través de dos formas de llevar esta cuenta. El hardware implementa solo una de estas técnicas:

1. Hora de uso

- Cada vez que la CPU realiza un direccionamiento incrementa un contador interno.
- A cada página se le asocia un registro de instante de uso y se añade al procesador un contador (reloj lógico) y cada vez que el procesador accede a esa página, copia el contador interno en el registro de la página.
- De esta manera siempre se tiene el "tiempo" de la última referencia a la página y se busca para sustituir la página con menor valor del contador.

Ventajas:

1. Fácil de implementar
2. Hardware poco costoso.

Desventajas:

1. - Cuando hay un overflow del contador interno del procesador, el algoritmo falla.
2. - Requiere una búsqueda en la tabla de páginas para encontrar la página menos usada recientemente.
3. - Se debe mantener los tiempos cuando se cambian las tablas de páginas (por la planificación de la CPU).

2. Pila

- La tabla de páginas en memoria central se mantiene ordenada usando un stack. Cada vez que se referencia a una página, se la pone en el tope de la pila.
- En la parte superior del stack siempre se encuentra la página más recientemente usada, y en la parte inferior, la menos recientemente usada.
- La página del fondo de la pila es la candidata a ser sustituida.
- Puesto que las entradas deben sacarse de la mitad de la pila, puede implantarse mejor con una lista doblemente enlazada, con apuntadores al inicio a al final. **Tiene como** ventaja **que n**o hay que buscar para efectuar el reemplazo y como desventaja que cada actualización es un poco más costosa

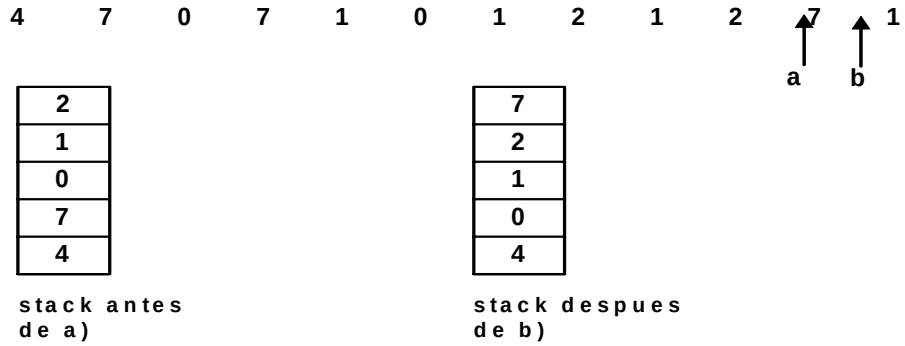


Figura 5.42 Uso de un stack

Desventajas de LRU:

1. La implantación de este algoritmo no podría concebirse sin la ayuda del hardware.
2. La actualización de los registros de reloj o pila debe efectuarse para *cada* referencia a memoria.
3. Mediante interrupciones para cada referencia (uso de software), sería diez veces más lenta cada referencia a memoria, haciendo más lento en la misma proporción a los procesos de cada usuario. Pocos sistemas tolerarían tal procesamiento adicional para administrar la memoria.

Ejemplo del uso del algoritmo LRU:

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	Serie de referencias
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	
	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	MARCOS
		7	0	1	2	2	3	0	4	2	2	0	3	3	1	2	0	1	7	
			F		F		F	F	F	F			F		F		F			FALLOS DE PÁGINAS

Fig. 5.43 Algoritmo de reemplazo de Páginas LRU

Se producirán 9 fallas de páginas con los 3 frames asignados y dicha traza de ejecución.

d). Otros algoritmos de Aproximaciones a LRU

Pocos sistemas proporcionan suficiente ayuda del hardware para un verdadero reemplazo de páginas LRU. Muchos sistemas ofrecen un *bit de referencia.* El hardware coloca a uno ("1") el bit de referencia en cada entrada de la tabla de páginas y cada vez que se hace una referencia a ella. Después de cierto tiempo, examinando estos bits, se puede determinar cuáles páginas se han usado, aunque no sabemos el orden de uso. Esta información adicional lleva a varios algoritmos que se aproximan a LRU.

1. NUR: Not Used Recently

Usa un bit asociado a cada entrada en la MPT llamado bit de referencia (reference bit), que la CPU pone en 1 cada vez que se accede a esa página.

El S.O. limpia periódicamente esos bits.

Las páginas candidatas a ser sustituidas serán las que tengan el bit de referencia en 0.

2) Additional-Reference-Bits Algorithm

Asociado con cada página hay un byte en el cual se van a almacenar los reference bits.

Este byte actúa como una historia de las referencias a la página.

Cada tanto, el S.O. pone el reference bit de la página en el high-order bit del byte de referencia y mueve los otros bits a la derecha.
1 ⇒ 01001100 ⇒ **1**0100110
(el último 0 fue descartado)
Las páginas candidatas a ser sustituidas serán las que tengan el reference Byte mas chico. En el caso extremo, el número puede reducirse a cero, dejando únicamente el bit de referencia.

3) Second-Chance Algorithm

FIFO modificado con un reference bit.

Si una página es seleccionada como víctima, primero se verifica su reference bit. Si es 0, se la reemplaza. Si es 1 se le da una segunda oportunidad, se limpia su reference bit y se selecciona otra página en orden FIFO.
A la página que se le brinda una segunda oportunidad se le establece como momento de llegada el momento actual. De esta manera, esta página no se va a reemplazar hasta que todas las demás páginas sean reemplazadas. Si una página se utiliza con frecuencia suficiente para mantener en 1 su bit de referencia, nunca será reemplazada.

4) Algoritmo del reloj

Este algoritmo básicamente es igual al de segunda oportunidad solo que con una implementación distinta. Muchas veces la lista de páginas candidatas a ser sustituidas se mantiene en una cola circular, en esos casos el algoritmo se lo conoce como **clock algorithm**
Este algoritmo degenera en FIFO si todas las páginas de la lista tienen su bit de referencia en 1.
Generalmente no se reemplazan páginas que han sido modificadas, para reducir el número de E/S

En este caso se tiene una lista circular, donde existe un puntero que apunta a la página mas antigua, luego cuando ocurre un fallo de página se consulta por el bit R de la página, si es 0 entonces esa página se elige para el reemplazo, en cambio si el bit R es igual a 1 entonces el bit R se pone en 0 y se incrementa el puntero de manera que apunte al siguiente nodo de la lista que corresponde a la siguiente página mas antigua.

El **algoritmo de reloj** puede hacerse más potente incrementando el número de bit empleado. En todos los procesadores que ofrecen paginación, se asocia un bit de modificación con cada página de memoria central y, por lo tanto, con cada marco. Este bit es necesario para que, cuando se modifica una página, no se reemplace hasta volverla a escribir en memoria secundaria. Es posible aprovechar este bit en el algoritmo de reloj del siguiente modo. Si se tiene en cuenta el bit de uso y de modificación, cada marco estará en una de las siguientes categorías:
(R=0;*M*=0) No accedida recientemente, no modificada.
(R=1;*M*=0) Accedida recientemente, no modificada.
(R=0;*M*=1) No accedida recientemente, modificada.
(*R*=1,M=1) Accedida recientemente, modificada.

El algoritmo de reloj se comporta de la siguiente forma:

1. comenzando en la posición actual del puntero, recorrer el buffer de marcos. Durante este recorrido, no cambiar el bit de uso. Se selecciona para el reemplazo el primer marco encontrado con R=0 y M=0.
2. si falla el paso 1, recorrer de nuevo buscando el marco con R=0 y M=1. Se selecciona para reemplazar el primer marco encontrado. Durante este recorrido, se pone a 0 el bit de uso de cada marco por el que se pasa.
3. si falla el paso 2, el puntero habrá retornado a su posición original y todos los marcos del conjunto tendrán el bit de uso a 0. Repetir el paso 1. Esta vez, se encontrará un marco para reemplazar.

La ventaja de este algoritmo sobre el algoritmo simple de reloj es que las páginas que no han sido cambiadas tienen preferencias para reemplazarse. Puesto que una página modificada debe escribirse al disco antes de ser reemplazada, se produce un ahorro de tiempo evidente.

5) Algoritmo del reloj con dos manecillas

El algoritmo aquí es igual al anterior solo que existen dos punteros un puntero va delante de otro, el primer puntero verifica el bit R de la página, si este es igual a 1 se pone en 0, luego cuando pase el segundo puntero verifica el estado del bit R si es 0 entonces intercambia la página sino lo pone en 0 y apunta a siguiente página. El objetivo de este algoritmo es privilegiar a aquellas páginas mas referenciadas para que se queden en memoria.

Otra variante del algoritmo del reloj es definir clases en lugar de bits.

El algoritmo funciona a un conjunto de clases que se definen de acuerdo a los valores que toman los bits R y M. La clasificación en distintas clases se presenta a continuación:

- ✓ Clase 0: R=0 y M=0 , no se ha hecho referencia a la página ni ha sido modificada.
- ✓ Clase1: R=0 y M=1, no se ha hecho referencia a la página, pero si ha sido modificada.
- ✓ Clase 2: R=1 y M=0, se ha hecho referencia a la clase, pero no se ha modificado.
- ✓ Clase 3: R=1 y M=1, se ha referenciado y se ha modificado la página.

Cuando ocurre un fallo de página el algoritmo contempla reemplazar aquellas páginas que pertenecen a la clase 0, 1, 2 y 3 en ese orden, es decir, si existen páginas de la clase 0 ellas son las que se reemplazan, si no existen páginas de esa clase se reemplazan páginas de la clase 1, si no hay de la clase 1 se reemplazan las de la clase 2, y así, sucesivamente.

6. LFU Least Frequently Used

Mantiene un contador por cada página del número de veces que fue accedida.
El contador se incrementa con cada acceso a la página.
La página con el contador más chico se sustituye.
Sufre un problema con páginas que son usadas muy intensamente al iniciar el programa y luego se dejan de usar. Se puede resolver moviendo los bits del contador 1 posición a la derecha a intervalos regulares de forma tal que el contador comience a achicarse.
Permanece en memoria aunque ya no se necesite.
No es muy utilizada porque su implementación es cara en términos de Hardware y no aproxima muy bien a OPT.

7) MFU Most frequently used

Reemplaza la menos recientemente usada.
Se basa en el argumento de que la página con el menor recuento probablemente acaba de llegar y aún tiene que usarse.
La implantación es costosa y no aproxima bien a OPT.

8) Enhanced Second-Chance Algorithm (Mejorado)

El algoritmo de segunda chance descripto puede ser mejorado considerando el bit de referencia y de modificación como un par ordenado. Con éstos dos bits se presentan las siguientes cuatro combinaciones:

- ♦ **(0,0)** ni usada ni modificada recientemente - mejor página para reemplazar.
- ♦ **(0,1)** no usada recientemente pero modificada - no es la mejor página a reemplazar, porque la página deberá escribirse para luego reemplazarse.
- ♦ **(1.0)** usada recientemente pero no modificada - seguramente será usada nuevamente pronto.
- ♦ **(1,1)** recientemente usada y modificada - probablemente será usada nuevamente y será necesario escribirla para reemplazarla.

Se le da importancia a las páginas que han sido modificadas para reducir el número de E/S requerida.

Cuadro comparativo de Algoritmos de reemplazo de páginas en memoria virtual, comparación de ventajas y desventajas.

Algoritmo	Descripción	Ventajas	Desventajas
Optima	Se elige para reemplazar a la página que va a estar mas tiempo en espera hasta que se produzca la próxima referencia.	Genera el menor número de fallos de página.	Es Imposible de implementar puesto que el sistema operativo tendría que conocer de antemano cuales van a ser las referencias futuras a las distintas páginas.
LRU : Least recently used (la menos reciente-mente Usada)	Se reemplaza la página de memoria que no se usa desde hace mas tiempo.	Tiene casi el mismo rendimiento que la política óptima en lo que respecta a fallos de página.	Overhead excesivo debido a lo complejo de implementación de este algoritmo.
FIFO (First in first out) Primero en entrar primero en salir.	Mediante un puntero que circula a través de los marcos del proceso en forma circular se van eligiendo las páginas a ser reemplazados.	Sencillez de implementación.	Rendimiento pobre.
Política del reloj	Se usa un puntero circular	Mayor rendimiento que	No alcanza el grado de

	con el agregado de un bit de uso, el cual se pone en 1 cada ves que se usa la página. Para que una página sea descargada de memoria el bit de uso tiene que estar en 0, si esta en 1 se lo pone en 0 y se sigue buscando circularmente hasta encontrar una página con el bit de uso en 0.	FIFO sin tener que usar un algoritmo tan complejo como LRU.	eficiencia que LRU.
Almacenamiento intermedio de páginas	Se crean dos listas de páginas a reemplazar: páginas no modificadas y páginas modificadas. Cuando se necesita reemplazar una página se elige primero de la lista de páginas no modificadas. Al quedar en memoria las páginas esto actúa como una caché.	Menor costo de reemplazo ya que las páginas no modificadas no son necesarias reescribirlas al disco.	
Políticas de reemplazo y tamaño de caché.	Se emplea una caché para el almacenamiento de páginas.	Mayor rapidez en ubicar una página para ser reemplazada.	

Tabla 5.4. Resumen de las principales características de los algoritmos de gestión de la memoria con swapping**.**

5.4.5. Consideraciones Sobre Administración de Memoria Virtual (Paginación por demanda)

a) La Curva de Swapping:

El uso del algoritmo LRU implica inspeccionar una pila, se puede desarrollar una curva de swapping en base al número de páginas residentes como un conjunto de working set y este queda determinado por el tamaño del de la página λ.

La cantidad de fallas de página pueden ser determinadas para cada working set y se puede graficar mediante un histograma de las ocurrencias y de las "distancias" de recuperación de página. La "distancia" es la profundidad en que se encuentra la que la página en la pila cuando es referenciada.

Se define como **Cadena de distancias** a la cadena formada por las respectivas distancias de recuperación de las páginas que son solicitadas para la ejecución de una tarea. La cadena de distancias tiene la misma cantidad de elementos que la cadena de referencias.

b) Función de Permanencia de la página en memoria

En las discusiones sobre políticas de reemplazo y el algoritmo de pila LRU, la implicancia de las fallas de página se puede ver en las siguientes curvas

Al observar el valor de la curva se refiere frecuentemente a la variación de una función F(m), donde m = número de marcos de página y su probabilidad de falla asociada. La curva desciende con el aumento del número de los marcos. Después desciende a medida que aumenta la disponibilidad de frames, nivela en un rango o región y después desciende nuevamente.

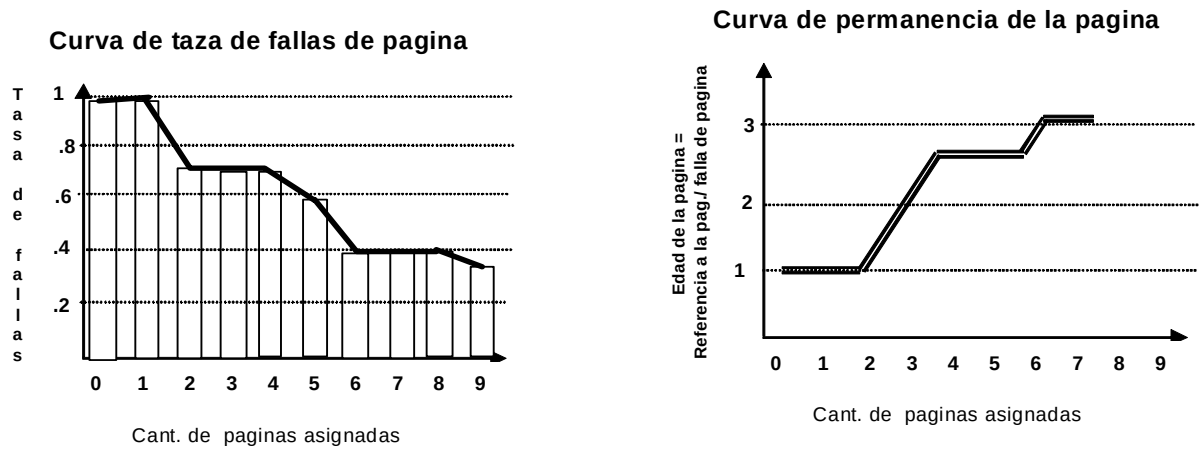


Fig. 5. 44 Comportamiento de las páginas en memoria.

La curva conocido como la Curva de Permanencia es el inversa del valor curva de falla de páginas (derecha arriba). Esta curva se refiere como L(m) y representa el promedio de "vida" de una página en la memoria central o sea su permanencia. En realidad es, el número promedio de referencias de páginas que ocurren en el proceso antes de que una página sea reemplazada. Esta curva es generada por una traza específica de referencias en un dado momento de ejecución y sirve solo al efecto de ver un determinado comportamiento del algoritmo LRU.

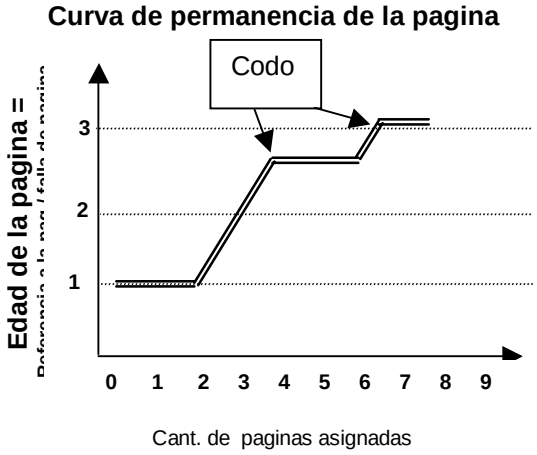


Fig 5.45 Curva de permanencia en memoria de la página

Si el número de páginas asignado al proceso es aproximadamente el tamaño del conjunto de localidad (working set), la vida de una página extiende a la vida del proceso y no fomentará su aumento. Como el diagrama muestra, cuando los marcos son suficientes para las páginas que se asignan al proceso que están siendo usadas por el proceso sobre el intervalo de tiempo inspeccionado, entonces la curva de permanencia permanece constante. En cambio donde la curva se incrementa para lograr un nuevo nivel constante se llama codo. Un proceso puede tener más de un working set por lo tanto más de un codo. El codo más importante es el primario, definido como el punto que aumenta al máximo valor de la relación de las páginas asignado a la permanencia y se asocia este punto con el desempeño óptimo.

c) Tasa de falla de página - Page Fault Rate (PFR)

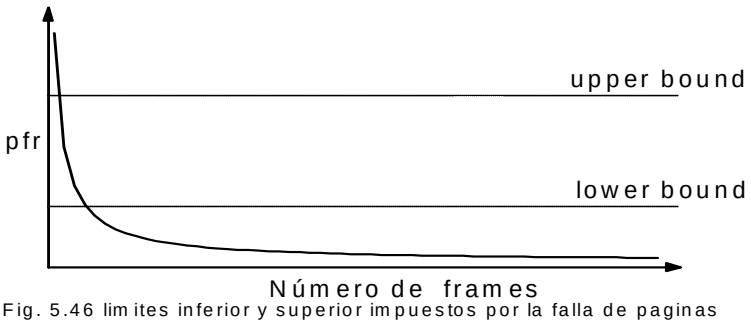


Fig. 5.46 limites inferior y superior impuestos por la falla de paginas

El S.O. incrementa un contador por cada page fault que provoca el proceso y se establece a priori un límite superior (upper bound) y un límite inferior (lower bound).

- Si pfr > upper bound ⇒ El S.O. aumenta el número de frames asignados al proceso.
- Si pfr < lower bound ⇒ El S.O. disminuye el número de frames asignados al proceso.

Si aumenta la tasa de page fault y no hay frames disponibles, se debe seleccionar un proceso y suspenderlo. Los frames liberados se distribuyen entre los procesos con tasas elevadas de page fault.

d) Buffering de páginas

Cumple una función similar a la de una memoria caché. Se utilizan listas de page frames para guardar un número de páginas que han sido remplazadas, para no tener que acceder a disco si son pedidas poco tiempo después. También permite que las páginas modificas que se reemplazan, sean actualizadas en disco en grandes grupos. Se suele combinar con un algoritmo de reemplazo FIFO.

e) Políticas de reemplazo y tamaño de caché

El tamaño de la memoria central es cada vez más grande y la cercanía en las aplicaciones cada vez menor. Para comenzar, el tamaño de las cachés es cada vez mayor. Actualmente, son alternativas de diseño factibles grandes tamaños de caché incluso de varios megabytes. Con una caché grande, el reemplazo de páginas de memoria virtual puede tener un gran impacto en el rendimiento. Si el marco seleccionado para reemplazar está en la caché, entonces se pierde el bloque de caché, así como la página que contiene.
En sistemas que emplean alguna forma de almacenamiento intermedio de páginas es posible mejorar el rendimiento de la caché sustituyendo la política de reemplazo de páginas por una política de ubicación de páginas en el buffer de páginas.

f) Almacenamiento intermedio de páginas

A pesar de que las políticas LRU y del reloj son superiores a la FIFO, ambas poseen una complejidad y sobrecarga que ésta última no padece. Sobre todo en el caso de las Memorias Centrales grandes (que superen los 256 MeBy)
Una estrategia interesante, que puede mejorar el rendimiento de la paginación y permitir el uso de una política de reemplazo de páginas más sencillas, es el almacenamiento intermedio de páginas. El algoritmo de reemplazo de páginas es FIFO. Para mejorar el rendimiento, no se pierde la pista de la página reemplazada, sino que se asigna a una de las dos listas siguientes: la lista de páginas libres, si la página no ha sido modificada o la lista de páginas modificadas, si lo ha sido.

g) Tamaño del conjunto residente

El sistema operativo debe decidir cuántas páginas traer, es decir, cuánta memoria central asignar a un determinado proceso. Aquí entran en juego varios factores:
Cuanto menor es la cantidad de memoria asignada a un proceso, mayor es el número de procesos que pueden estar en memoria central en cualquier instante.
Si en memoria central hay un número relativamente pequeño de páginas de un proceso entonces, a pesar del principio de cercanía, el porcentaje de fallos de página será algo mayor.

h) Alcance del Reemplazo

El alcance de un reemplazo puede clasificarse en global o local. Ambos tipos de políticas son activadas por un fallo de página que se produce cuando no hay marcos libres. Para seleccionar la página a reemplazar, una política de reemplazo local escoge únicamente de entre las páginas residentes del proceso que originó el fallo de página. Una política de reemplazo global considera todas las páginas en memoria como candidatas para reemplazar, independientemente del proceso al que pertenezcan.

i) Políticas de Vaciado

Una política de vaciado es la contraria a una política de lectura; se preocupa de determinar el momento en que hay que escribir en memoria secundaria una página modificada. Las dos alternativas más habituales son el vaciado por demanda y el vaciado previo.
Con vaciado por demanda, una página se escribirá en memoria secundaria sólo cuando haya sido elegida para reemplazarse.
Una política de vaciado previo escribe las páginas modificadas antes de que se necesiten sus marcos, de forma que las páginas puedan escribirse por lotes.

j) Control de carga

El control de carga consiste en determinar el número de procesos que pueden estar en memoria central, lo que ha venido en llamarse grado de multiprogramación. La política de control de carga es crítica para la efectividad de la gestión de memoria.

k) Grado de multiprogramación

Cuando el grado de multiprogramación supera un pequeño valor, se podría esperar que la utilización del procesador aumentará, puesto que hay menos posibilidades de que todos los procesos estén bloqueados. Sin embargo, se alcanza un punto en el que el conjunto residente en promedio no es adecuado. En este punto, el número de fallos de página se eleva drásticamente y la utilización del procesador se desploma. Entra en Thrashing.

Hay varias formas de abordar este problema. Los algoritmos del conjunto de trabajo o de frecuencia de fallos de página incorporan implícitamente el control de carga. Solo pueden ejecutar aquellos procesos cuyos conjuntos residentes sean suficientemente grandes. Dado un tamaño exigido para el conjunto residente de cada proceso activo, la política determina automáticamente y dinámicamente el número de programas activos.

l) Suspensión de procesos

Si se va a reducir el grado de multiprogramación, deben suspenderse uno o más procesos actualmente residentes. Carr[2] enumera las posibilidades siguientes:

1. Procesos con la prioridad más baja: Esta alternativa toma una decisión de política de planificación y no tiene que ver con las cuestiones de rendimiento.
2. Procesos con fallos de página: El razonamiento es que hay una gran probabilidad de que las tareas que provocan fallos no tengan residente su conjunto de trabajo y, suspendiéndolas, el rendimiento pierda lo menos posible.
3. Último proceso activado: Este es el proceso con menos posibilidades de tener su conjunto de trabajo residente.
4. Proceso con el conjunto residente más pequeño: Este es el proceso que necesita el menor esfuerzo futuro para volver a cargar el conjunto residente.
5. El proceso mayor: Esta alternativa obtiene la mayor cantidad de marcos libres en una memoria muy ocupada, haciendo poco probables más desactivaciones en breve.

5.4.6. SISTEMAS MIXTOS: SEGMENTACIÓN CON PAGINACIÓN POR DEMANDA.

- Combinan las técnicas de Segmentación y de paginación.
- En general los segmentos tienen un tamaño múltiplo de páginas.
- No se necesitan tener cargadas en memoria central todas las Páginas de los segmentos.
- La dirección virtual se organiza en tres partes:

Nro. de Segmento	Nro. de Página	Offset (Desplazamiento)
s	p	o

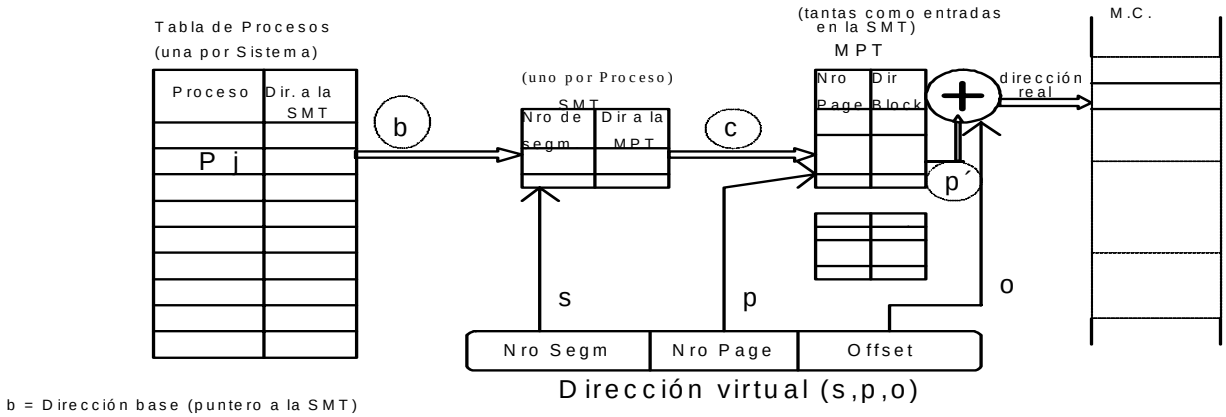
Fig. 5.47 Dirrección Virtual = (S,P,O)

La Traducción se realiza:

- Primero: Se efectúa una búsqueda asociativa en la tabla de segmento (SMT) y luego en la tabla de página (MPT) de ese segmento. Puede ocurrir:
  1. Que el segmento no está cargado (no existe el bit de presencia de segmentos en memoria central), entonces se produce una interrupción por falta de segmento en memoria central (***Segment fault***), luego se genera una búsqueda del segmento en el Disco, se crea la tabla SMT y la MPT para ese segmento. Si hay lugar se cargan las páginas necesarias y se reinicia le ejecución.
  2. Que la página referenciada no está cargada en memoria central por lo que se produce una interrupción de falla de página (***page fault***).
- Segundo: una vez localizada la correspondencia entre segmento y página en las tablas y hecho la suma del desplazamiento, se obtiene la dirección real donde está cargada en la memoria física.

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

[2] Carr, R. Virtual Memory Management. Ann Arbor. MI: UMI Research Press, 1984



b = Dirección base (puntero a la SMT)
s = Número de segmento (Dir. Virtual)
p = Número de Página (Dir. Virtual)
o = Offset (Desplazamiento dentro de la página)
p´= Puntero al Bloque de M.C. donde esta la Página.

M.C. = Memoria Central
SMT = Segment MemoryTable
MPT = Memory Page Table

Fig. 5.48 direccionamiento de segmentación con paginación por demanda

Ventajas:

1. Permite compartir segmentos.
2. No es necesario cargar la totalidad de los segmentos en memoria central ni la totalidad de las Páginas. Solo lo que se necesite.
3. No se requiere compactación.

Desventajas:

1. Requiere más Hardware para el direccionamiento.
2. Es más lento en la ejecución (por el mecanismo de traducción de las direcciones virtuales)
3. El S.O. ocupa más Memoria.
4. Aumenta la fragmentación interna.

a) Términos de Memoria

Address space Id: Para el control de las páginas cargadas en la memoria, se cuenta con un conjunto de tablas: Tablas de frames, Tablas de páginas y Mapas de disco. Las tablas de página contienen información sobre las páginas cargadas en memoria central. Pero en el caso de que una página no se encuentre en memoria central y sea solicitada para continuar la ejecución de un proceso el mapeador de direcciones (maper) llamará a la interrupción de página no presente (page fault). Esta página debe ser cargada desde la memoria de disco a la memoria central. Para eso se cuenta con el mapa de disco que contiene la dirección en la que se encuentra la página en el disco. Es posible combinar tablas de páginas y el mapa de disco y así formar una sola tabla.

Algoritmos de pila: Existen algoritmos que pueden representarse mediante la utilización de una estructura de datos llamada pila. Estos algoritmos (tales como LRU, sus variantes, etc.) son llamados algoritmos de pila.

Anomalías de las políticas de reemplazo: En cuanto a la política FIFO, su principal desventaja es que, si una página es cargada en un principio porque en ella se encuentra una variable que será utilizada muchas veces en el programa, entonces el reemplazarla significaría tener que volver a cargarla nuevamente lo que es realmente poco eficiente. Un caso particular, es que suelen generarse más page fault cuando hay mas páginas cargadas en memoria. Esta anomalía fue demostrada por Belady y es conocida como la **Anomalía de Belady**. La política LRU tiene como principal desventaja que es bastante complicada de implementar y requiere de mucha información que hace que las tablas de página crezcan demasiado y causando altos niveles de overhead.

Asignación de almacenamiento automática: Como es sabido, un proceso, debe ser ejecutado desde memoria central. Si bien las cantidades de memoria que se manejan en los sistemas actualmente es enorme y su costo cada vez menor, es posible que la memoria sea utilizada de una manera mas eficiente si solo se cargan las partes esenciales para la ejecución de un proceso, almacenando las áreas de memoria que serán utilizadas mas adelante en un medio de almacenamiento secundario tal como un disco. De esta manera, la ejecución del proceso se detendrá cuando ocurra un page fault. Ese page fault será atendido y se hará uso de los identificadores de campo de dirección (mas precisamente del mapa de disco) para traer la página requerida a memoria central y que continúe la ejecución del proceso que se había bloqueado.

Asignación dinámica de memoria: A diferencia de la asignación estática de memoria, la memoria a ser utilizada varía en cuanto a las necesidades del programa. Esta memoria es brindada en tiempo de ejecución y debe ser liberada al finalizar la misma.
Asignación estática de memoria: Método de asignación de memoria en el cual se asume que la disponibilidad de memoria puede conocerse con anticipación (antes de la ejecución).
Bit de modificación. Indica si una partición de memoria ha sido modificada durante su estadía en la memoria.
Bit válido-inválido. Indica la presencia de una página en la memoria central.
Bloques – marcos. En este esquema, la memoria física es dividida conceptualmente en porciones de tamaño fijo que se denominan marcos de página.
Buddy system: Método de particionado de memoria en el que se la divide sucesivamente en dos hasta encontrar un bloque en el que quepa el proceso que requiere ser albergado en memoria. De esta manera, la memoria queda organizada de a pares (compañeros) de igual capacidad que están listos para recibir a otro proceso que requiera ejecución.
Cadenas de referencia: Se llama cadena de referencia a la secuencia de páginas que son referenciadas en la ejecución de un proceso.
Cadena de distancias: Se llama cadena de distancias a la cadena formada por las respectivas distancias de recuperación de las páginas que son solicitadas para la ejecución de una tarea. La cadena de distancias tiene la misma cantidad de elementos que la cadena de referencias.
Carga del sistema: El numero de procesos activos en el sistema.
Código reentrante. Éste no será modificado durante el tiempo de ejecución.
Cola de memoria: Cola en la que se mantienen las transacciones hasta que los recursos que estas requieren estén disponibles.
Compactación. De tanto en tanto, el S.O. mueve los procesos para que estos queden contiguos en la memoria y así dejar un solo hueco de memoria disponible.
Conjunto activo: Se define Active set al conjunto de procesos que están siendo ejecutados en un instante dado.
Conjunto de localidades: En el comportamiento de un programa se pueden observar diferentes fases o periodos en los que cambian los contenidos de la memoria central. El tiempo entre este cambio de fases se denomina transición. Se denomina conjunto de localidades al conjunto de páginas que serán utilizadas en una determinada fase.
Desplazamiento. Cuánto hay que moverse desde la base de una partición de memoria.
Dirección física. La dirección real de las posiciones de memoria.
Dirección lógica: La dirección que brinda el proceso. Dirección que consta de el número lógico de página y la ubicación del dato dentro de esta. La dirección será tomada por el mapper que se fijara si esta en memoria central mediante la utilización de las tablas de páginas. En caso de estar, la dirección lógica será traducida a física y se accederá al dato; y en caso de no estar, el mapper emitirá la señal de page fault que bloqueara el proceso y traerá desde la memoria de intercambio a la memoria central.
Las direcciones lógicas son las que se utilizan en el llamado Espacio lógico (o virtual). Este espacio es el utilizado por los programadores de aplicaciones. Esta dirección cuenta con dos campos:

SELECTOR *: DESPLAZAMIENTO*

El selector es el campo que determina la posición de un segmento. El selector es un campo cuyo valor se suma a la base del segmento para hallar la dirección a acceder.

Cuando se tiene la dirección lógica, es necesario efectuar una serie de traducciones para que esa dirección haga referencia a una dirección física en la memoria.

Distancia de recuperación de página: El algoritmo LRU (Last Recently Used) puede ser visto como una pila de páginas residentes en memoria en la que la primer posición de la pila es la página que ha sido recientemente usada. Se llama distancia de recuperación de página a la distancia que hay que recorrer (desde el principio hasta el final) en la pila hasta conseguir la página pedida. De esta manera, si se solicita la página numero 1 y esta se encuentra en la cima, entonces la distancia es 1; si se pidiera luego la página 4 y esta estuviese en la segunda posición de la pila, entonces la distancia sería 2.
Fallo de página. Cuando se intenta acceder a una página que no se encuentra en la memoria central.
Formato little endian: Administración de datos en la memoria en la que el byte de menos peso se ubicara en la dirección de memoria más baja. Es la notación seguida en el Intel 486 DX.

Fragmentación externa: A medida que pasa el tiempo, la memoria se fragmenta cada vez más, y la utilización de la misma se vuelve menos eficiente. Cuando se le asigna a cada proceso el monto de memoria que precisa, existe la posibilidad de que cuando un proceso desocupe un bloque de memoria, este sea ocupado por otro proceso mas pequeño que el que antes estaba en esa zona y, así, deje fragmentos de memoria demasiado pequeños para ser usados por otro proceso. Este

fenómeno es llamado fragmentación externa.

Fragmentación interna: Ésta consiste en la parte de una página que no es aprovechada. Cuando un proceso es mas pequeño que la partición que se le ha sido asignada, este proceso desaprovecha un trozo de memoria que podría ser asignado a otro proceso un lugar de ser retenido por este.
Grado de Multiprogramación: Se entiende por multiprogramación a la ejecución de múltiples procesos en un sistema mono-procesador. La multiprogramación aprovecha al máximo el uso de CPU dependiendo del grado en el que se la utilice. A mayor grado de multiprogramación, mayor cantidad de procesos en simultáneo, por lo tanto mayor uso de CPU.
Hiperpaginación. Consume más tiempo páginando que procesando.
Mapa de disco: Indica en que parte del disco esta ubicada cada página. Es una parte de los identificadores del espacio de direcciones.
Mapa de páginas. Describe los bloques de memoria central que están siendo usados por el proceso.
Mejor ajuste. Busca el área más chica que pueda contener el requerimiento. Produce áreas libres muy pequeñas.
Mapper: El mapper es la parte del sistema operativo que se encarga de traducir las direcciones lógicas en direcciones físicas (que son las direcciones reales en donde se encuentran los datos en la memoria). Para llevar a cavo esta función, cuenta con un conjunto de tablas asociadas a las páginas. Es necesario que la función de traducción del mapper sea rápida por lo que este cuenta con una memoria asociativa.
Memoria Asociativa: La traducción de direcciones implica una gran cantidad de accesos a memoria. Para evitar estos excesivos accesos, se implementa una memoria ultrarápida y de tipo asociativa. Esta memoria se denomina TLB.
Memoria dedicada. El programador posee acceso absoluto a toda la memoria central y es el encargado de la protección de la misma.
Memoria virtual: Técnica para simular más memoria central de la que dispone el sistema. Dado que los datos y el código de un proceso deben estar residentes en memoria central para su ejecución, es posible que la memoria que le ha sido asignada a ese proceso no sea suficiente para alojar a todo el código y los datos. Por esta razón se utiliza la memoria virtual. Solo se carga lo extremadamente necesario para el comienzo de la ejecución del programa, y el resto puede residir en un medio de almacenamiento secundario (Como por ejemplo un disco). Cuando se necesite algo que no esta cargado en memoria central el sistema operativo bloqueara al proceso y traerá de la memoria secundaria a la central el dato que se necesite.
Menos frecuentemente usado: Política de reemplazo en la que se sustituye a la página que es usada menos a menudo.
Menos recientemente usado (LRU): Política de reemplazo en la que se sustituye la página que hace más tiempo que no es referenciada. Se utiliza el principio de localidad, ya que las páginas que no son referenciadas hace un tiempo considerable tienen menos probabilidades de volver a ser referenciadas que las páginas que se traerán de memoria actualmente. Posee la importante desventaja de posee un costo de implementación muy alto.
Page frame: Se llama Page Frame a los trozos de memoria que a los que son asignadas las páginas.
Page fault: Interrupción que es disparada por el mapper cada vez que quiere hacerse referencia a una página que no se encuentra en memoria central.
Página. División de un programa en porciones de tamaño fijo de n bytes cada uno utilizado en una administración de la memoria llamada paginación.
Paginación: Procedimiento de gestión de memoria en el que se dividen y manipulan a los programas en trozos de tamaño fijo llamados páginas. El tamaño de las páginas puede variar dependiendo de la arquitectura. Es un sistema muy eficaz utilizado por la mayoría de los sistemas operativos multitarea. La principal ventaja de la paginación es el traslado de información desde la memoria virtual a la física, puesto que al poseer las páginas un tamaño fijo, los algoritmos para el movimiento de bloques de memoria con este sistema son mucho más sencillos. La ejecución de un programa puede comenzar sin que estén cargadas todas sus páginas en la memoria central. Para manejar el estado de las páginas el sistema cuenta con una tabla de páginas por cada proceso. Es muy común que la paginación se combine con la segmentación para así, hacer un uso más eficiente de la memoria.
Paginación por demanda (Demand Paging): Paginación con swapping. Es una política de búsqueda de páginas. Con la paginación por demanda, la página será traída a memoria solo cuando se haga referencia a un objeto de esa página. Al comienzo de la ejecución de un proceso este tendrá varias page faults que serán atendidos y la/s página/s serán traídas a memoria central. Ahora, teniendo en cuenta el principio de localidad, podemos asegurar que la mayoría de las referencias futuras no causarán page fault por encontrarse en las páginas recientemente cargadas.

Paginación de multinivel. Las tablas de páginas a su vez se encuentran paginadas.
Particiones de memoria: Manera en que se divide la memoria. Puede ser fija o variable.
Particiones fijas. La memoria se encuentra dividida en particiones de tamaño fijo, y cada una de ellas contiene un proceso.
Peor ajuste. Asigna el bloque más grande posible. Al producir el área sobrante más grande, ésta puede ser más útil que un área más pequeña.
Prepaginación. La prepaginación es traer páginas a la memoria para prevenir la ocurrencia de fallos de página.
Primer ajuste. Estrategia para realizar la asignación, en la cual recorre la tabla hasta encontrar el primer área cuya capacidad alcance, comenzando del principio de la tabla o bien de donde terminó su última búsqueda.
Primero en entrar, primero en salir (FIFO): Política de reemplazo en la que se sustituye primero la página que hace más tiempo que se encuentra en memoria. Es una política bastante utilizada, fácil de entender y de implementar pero es poco eficiente en algunas circunstancias.
Principio de localidad: El principio de localidad afirma que las referencias a datos y a código de un proceso tienden a estar juntos, por lo que instrucciones cercanas en el código también estarán cerca (muy probablemente dentro de la misma página) cuando sean cargadas en memoria. Con este principio se pueden hacer predicciones muy importantes a cerca del comportamiento de un proceso.
Política de reemplazo mínimo u óptimo (MIN u **OPT):** Política de reemplazo en la que se sustituirá la página que tenga un mayor tiempo hasta una próxima referencia. Este algoritmo es imposible de implementar porque requiere que el sistema operativo tenga un completo conocimiento de lo que ocurrirá en un futuro. Es el algoritmo que brinda menos situaciones de page fault.
Proporción de falla: Probabilidad de que ocurra un page fault
Política de reemplazo: Como es el sistema operativo el encargado del acarreo de páginas desde la memoria virtual a la física, este también debe decidir que página reemplazar cuando se debe ubicar en memoria una nueva. Para esto, existen varias políticas tales como LRU, FIFO, MIN, etc.
Pool de frames: Dentro de las tablas de páginas, se llama pool de frames a las filas que poseen los flags P y M en “0”. Son usados para satisfacer page faults.
Protección. Evitar que procesos ajenos accedan a información en forma accidental o intencionada.
Queueing network model: Este modelo representa como un sistema con una configuración especifica procesa los trabajos. Este modelo varia según la complejidad de la configuración del sistema. Se puede evaluar el rendimiento del sistema (throughput, tiempo de respuesta, uso del CPU) utilizando parámetros como: Tiempo de servicio, visit ratio, carga del sistema, etc.
Registro Base. Uno de los registros marca el límite inferior de una partición de memoria.
Registro límite. Hasta dónde se puede desplazar dentro de la partición.
Reubicabilidad. Cargar y descargar (swap in – swap out) de la memoria a los procesos si estos están activos o inactivos respectivamente.
Segmento. División de la memoria agrupando lógica y semánticamente los datos.
Segmentación: Es un procedimiento de gestión de memoria que divide al espacio de un programa en bloques de longitud variable llamados segmentos. Los segmentos responden a la estructura lógica del programa. Para la administración de los segmentos, el sistema cuenta con los “descriptores de segmento”. Estos descriptores son estructuras compuestas por “x” bytes que poseen una base, un limite de “y” bits que son utilizados para los atributos del segmento (tales como: presencia, nivel de privilegio, accedido, granularidad, etc.). La desventaja principal de este modo de particionar la memoria es que, al ser bloque de longitud variable, aumenta la complejidad de los algoritmos para el movimiento de información. Esta técnica puede ser implementada junto con la paginación.
Sustitución global de páginas. El S.O. pueden quitarle marcos a cualquier otro proceso.
Thrashing: Situación en la que el procesador pasa mas tiempo trayendo páginas a la memoria o llevando al disco, que ejecutando instrucciones.

Tabla de páginas: Estructuras que utiliza el sistema para traducir las páginas y conocer la ubicación del frame de cada página del proceso. Hay una tabla de páginas por proceso. La tabla cuenta con un campo de un bit llamado “U” (Used), un bit llamado “P” (Presence), otro bit llamado “M” (Modified) y un campo que contiene el frame en donde esta alojada la página (siempre y cuando el flag P esté en “1”) mas otros bits de protección. La tabla tiene una entrada por cada página del proceso

Tabla de páginas invertidas. Esta tabla es solamente una para todo el sistema, y contiene una entrada para cada página real de memoria, a partir de la cual se asocia la dirección virtual de la página almacenada en esa localidad.
Tabla de páginas multinivel: Cuando los procesos son muy grandes, la tabla de páginas puede requerir bastante espacio en memoria para ser alojada. Para evitar comprometer al proceso con el espacio que le fue asignado, muchos sistemas utilizan este tipo de tablas de páginas en las que la

dirección virtual de la página se particiona y la tabla de páginas se divide jerárquicamente. De esta manera, no es necesario que este la totalidad de la tabla de páginas cargada en memoria.
Tabla de frames: Es una tabla mantenida por el sistema que contiene información a cerca de cómo esta siendo utilizada la memoria central del sistema. Contiene un espacio para la dirección y otro para la página a que se encuentra en esa dirección.
Tabla de segmentos. Una tabla con una entrada por cada segmento que posea el proceso.
Tiempo de respuesta: El tiempo total de un trabajo en el sistema. Incluyendo el tiempo de servicio.
Tiempo de servicio: El tiempo que tarda el sistema en completar un trabajo en una tarea.
Throughput: El numero de transacciones completadas por el sistema o por un servidor por unidad de tiempo.

TLB: Dado que para la traducción de direcciones en la paginación se requisen dos accesos a memoria, lo que hace a este proceso bastante lento, se introdujo una memoria de tipo asociativa llamada TLB (Translation Lookside Buffer). La TLB guarda la traducción de direcciones lineales a físicas de las últimas páginas que se han manejado. Cuando la CPU requiere una página, consulta a la TLB y en caso de poseerla, se tardan solo 2 ns en acceder a ella. Si la página no se encuentra en la TLB, entonces se procede a traducir la dirección efectuando los dos accesos necesarios a memoria para actualizar el TLB y además con el retardo suplementario de 2 ns (que después de todo no es significativo teniendo en cuenta que solo se tardarán 2 ns en acceder a esta página la próxima vez que se la necesite).

Unidad de control: Es una parte del Modelo de administración de memoria (Queueing Network Model) y se encarga de manejar la entrada al sistema desde la cola de memoria.
Uso del procesador: Se define como uso de CPU (CPU Utilization) a la cantidad de tiempo en el que la CPU esta ocupada en la ejecución de un proceso y no esta esperando por dispositivos periféricos.
Ultimo en entrar primero en salir (LIFO): Política de reemplazo en la que la página a reemplazar será la última que ha sido cargada en memoria central.
Visit Ratio: Es el numero de veces que una transacción es atendida por un servidor antes de salir del sistema.
Working set model (Conjunto de trabajo): Se denomina Working set al conjunto de páginas que están siendo usadas. De esta manera, es posible rastrear el Working set para asegurarse de que las páginas a ser usadas se encuentren en memoria. Un sistema que actúa de esa manera es un sistema que utiliza Working Set Model.

5.5. Bibliografía recomendada para este Módulo

1. Operating Systems. Internals and design principles(4th Edition), Stallings Willams; Prentice Hall, Englewood Cliff, NJ. –2000- , 779 pages.

2. Operating Systems Concepts (Fifth Edition), Silberschatz, A. and Galvin P. B; Addison Wesley 1998, 850 Pages.

3. Operating Systems Concepts and Design (Second Edition), Milenkovic, Millan; Mc Graw Hill 1992

4. Modern Operating Systems, Tanenbaum, Andrew S.; Prentice - Hall International 1992, 720 pag.

5. Operating Systems, Design and Implementation, Tanenbaum, Andrew S.; Prentice - Hall International, 1987, 800 pag.

6. Fundamentals of Operating Systems, Lister, A.M. ; Macmillan, London 1979

7. Operating System Design - The XINU Approach, Comer, Douglas; Prentice Hall 1984

8. Operating System, Lorin, H., Deitel, H.M.; Addison Wesley; 1981;

9. A discipline of Programming, Dijkstra, Edward W.; Englewood Cliffs, Prentice Hall; 1976

10. The Art of computer Programming (Volumen 1, 2, y 3); Knuth, Donald; Addison - Wesley Publishing Co.; 1974

11. Operating Systems: Structures and Mechanisms; Jensen Philippe; Academic Press 1985

12. The UNIX Operating System, Christian, K.; John Wiley; 1983

13. UNIX System Architecture, Andleigh, Prabhat K.; Prentice - Hall International; 1990

14. The UNIX Programming Environment, Kernighan, B.W. and Pike, R.; Prentice - Hall International 1984

15. The UNIX System V Environment, Bourne, Stephen R.; Addison Wesley; 1987

16. UNIX Utilities a Programmerer's Guide.; Tare, R. S.; Mc Graw-Hill.; 640 pag.

17. Tricks of the UNIX Masters. ; Sage, R. S.;

18. UNIX System Readings and Applications. (Vol I y II); AT&T Bell Laboratories.; Prentice Hall Englewood Cliffs; 1987- 416 pag.

19. Introducing UNIX System V; Morgan, R. and McGilton, H.; Mc Graw Hill; 480 pag.

20. Sistemas Operativos Conceptos y diseños.(Segunda Edición); Milenkovic Milan; Mc Graw Hill; 1994.

21. Sistemas de Explotación de Computadores; CROCUS; Paraninfo; 1987 424 pag.

22. Sistemas Operativos MS-DOS, Unix, OS/2, MVS, VMS, OS/400; E. Alcalde,J. Morera, J.A. Pérez Campanero.; Mc Graw Hill; 1992.

GLOSARIO DE TÉRMINOS EN IDIOMA INGLÉS

Overhead	Stack	Push
Pop	Compile time	Run Time
Load Time	In core	Address binding
Overlays	Stub	Memory Management Unit
Swapping	Fence Register	Compile time binding
Run Time binding	Load Time binding	drivers
High memory	Job	Hole
First Fit	Best Fit	Worst Fit
Next Fit	Address	Size
Overflow	Frames	Offset
Memory Page Table	Page offset	Page number
Page Fault	Context switch	Tag
Hit Ratio	Read	Write
Execute	Trap	Trap Memory Protection
Page Table Length Register	Segment Memory Table	Segment Memory Table Base Register
Segment Memory Table Limit Register	Buddy System	Buddies
Swaped - out	Swap- in	Swap out
Merge	Thrashing	Pager
First Level Interrupt Handler	Sucess frecuency function	Least Recently Used
Not Used Recently	Additional reference bit algorithm	Reference bit
High order bit	Second Chance algorithm	Clock algorithm
Most frecuently used	Least Frecuently used	Enhanced Second Chance algorithm
Working Set	Working Set Window	Upper bound
Lower bound	Page Fault Ratio	Segment Fault.

Multiprogramming with a Fixed number of task Multiprogramming with a Variable number of task
Translation look-aside Buffer

GLOSARIO DE TÉRMINOS EN CASTELLANO

Tiempo de acceso	Ciclo de memoria	Funciones del administrador de memoria central
Objetivos del administrador de memoria central	Protección	Uso compartido de códigos y datos
Reubicabilidad	Códigos reentrantes	Tiempo de Compilación
Tiempo de ejecución	Tiempo de carga	Carga dinámica
Enlace dinámico	Overlays	Mapa de Memoria central real
Mapa de Memoria central virtual	Dirección física	Dirección lógica

Unidad de administrador de memoria central

Memoria dedicada
Registro Base
Primer ajuste
Ajuste rápido
Compactación
Bloques - Marcos
Número de Página
Porcentaje de aciertos
Tabla de Páginas invertidas
bit de modificación
Tabla de segmentos
Sistema compañero
Paginación por demanda

Técnicas de administración de memoria central

Asignación contigua simple
Particiones Fijas
Mejor ajuste
Fragmentación externa
Paginación
Mapa de Páginas
Desplazamiento
Páginas compartidas
Bit válido - inválido
Violación de protección de memoria
Segmento
Técnica de intercambio
Memoria virtual

Sistema embebido
Registro límite
Particiones Variables
Peor ajuste
Fragmentación interna
Página

Hardware del Administrador de Memoria central

Registros asociativos
Tabla de Páginas
Bit de referencia
Paginación de multinivel
Segmentos compartidos
Azotado o hiperpaginación
Falla de Página

Algoritmos de reemplazo de Páginas

Sustitución de Páginas
Sustitución Global de Páginas
Sustitución Local de Páginas
Anomalía de Belady
Algoritmos de pila
Localidad
Modelo de Working set

Algoritmo de asignación de frames

Trazas de referencias
Prepaginación

Sistemas mixtos de Segmentación con paginación por demanda.

ACRÓNIMOS USADOS EN ESTE MÓDULO

DMA	Direct Memory Access	dir	dirección
PCB	Process Contol Block	ALU	Aritmetic Logic Unit
E/S	Entrada / Salida	PGR	Programa
RAM	Random Access Memory	RL	Registro límite
ROM	Read Only Memory	RB	Registro base
CPU	Central Processing Unit	LS	Límite superior
M.C.	Memoria central	Px	Proceso x
S.O.	Sistema Operativo	FF	First Fit
R/W	Read / Write	BF	Best Fit
DASD	Direct Access Storage Device	WF	Worst Fit
M.M.U.	Memori Manager Unit	NF	Next Fit
FR	Fence Register	MPT	Memory page table
FIFO	First In First Out	MFT	Memory Frame table
TLB	Translation lookaside buffer	PTBR	Page Table Base Register
MIPS	Millones de Instrucciones por segundo	PTLR	Page Table Length Register
ABM	Altas Bajas Modificaciones	SMTBR	Segment Memory Table Base Register
tma	tiempo medio de acceso	SMTLR	Segment Memory Table Length Register
tpf	tiempo de page fault	OPT	Optimo
pf	probabilidad de ocurrencia de Page fault	LRU	Least Recently Used
MFU	Most frecuently used	NUR	Not Used Recently
WS	Working set	LFU	Least Frecuently Used
SMT	Segment Memory table	PFR	Page Fault Rate

Anexo 5.A.1: Conceptos sobre Memoria Central (M.C.)

Es común que a la memoria del computador se la denomine central o principal para diferenciarla de otro tipo de almacenamiento, por ejemplo los discos, cintas, etc.

5.A.1.1. Definiciones de conceptos utilizados en memorias.

Las memorias: son dispositivos que permiten almacenar la información (datos, resultados, programas, imágenes, señales, etc.) y recuperarlos cuando cualquier elemento del computador lo requiera.

Se denomina ***Punto de memoria (o celda***) al dispositivo que almacena un bit. Físicamente está representado por un biestable (Flip-Flop), o por un capacitor, una "Burbuja magnética", un núcleo de ferrite, etc.

El punto de memoria puede tomar solamente dos estados binarios, "1" o "0".

Byte equivale a un término binario (***B***inar***y te***rmin) generalmente compuesto por 8 bits.

Carácter: es el conjunto de bits necesarios para representar un símbolo de un código (alfabético, numérico, especial, etc.).

Palabra: es una unidad lógica de información. Generalmente está compuesto por un número entero de Bytes o caracteres.

El tiempo se mide en segundos. Los submultiplos mas utilizados en computación son:

Milisegundo(ms): milésimo de segundo = 10^{-3} segundos.

Microsengundo(μs): millonésimo de segundo = 10^{-6} segundos.

Nanosegundo(ns): mil millonésimo de segundo = 10^{-9} segundos.

Picosegundo (ps): billonésimo de segundo = 10^{-12} segundos.

Lectura de memoria: es el proceso de extracción (recuperación) de la información, que fue previamente almacenada en una determinada posición.

Escritura en memoria: es el proceso mediante el cual se registra (guarda) la información en una posición dada.

Dirección de memoria: es la ubicación física (posición) de la palabra en la memoria.

Direccionamiento: es la función usada para localizar una información dentro de la memoria.

Tiempo de acceso: es el tiempo que transcurre entre el instante en que se lanza una operación de lectura y el instante en que se dispone de la primera información buscada.

Capacidad de memoria: Es el número máximo de bits que puede contener la memoria.

La capacidad se expresa en unidades que son potencias de dos. Es binaria.

Es común referirse a la ***capacidad de almacenamiento*** que es la cantidad de caracteres, símbolos de un código o palabras de 8 bits que se puede contener una memoria en un sistema.

Las unidades convenidas son: el **KiBy** (***K***ilo***By***te) que representa a 2^{10} = 1024 Bytes, el **MeBy** (Megabyte que equivale a 10^6), el **GiBy** (GigaByte = 1000 MBy = 10^9), y el **TeBy** (Tera = 10^{12}). (Ej. 64KiBy, 128 KiBy, 256 KiBy, 512 KiBy, 1 MeBy, etc.)

Ciclo de memoria:

a) En el caso de núcleos magnéticos: la lectura es destructiva en estos dispositivos, y requiere de un proceso para regenerar y reescribir la información en cada lugar; por lo que esta operación se realiza en dos fases:
   a.1. grabación = puesta a cero + grabación
   a.2. lectura = lectura + regrabación.

   Al conjunto de ambas fases en cada caso, se lo llama "ciclo de memoria" que es algo mayor que el tiempo de acceso.

b) En los circuitos integrados: se toma el proceso de una lectura; (la escritura es semejante):

- El bus de direcciones tiene la información de la posición de la palabra de memoria a leer que le fuera enviado o por la CPU o por un Procesador DMA (Direct Memory Access).
- Esa dirección es físicamente descodificada por el controlador de la Memoria.
- Por el bus de control llega la orden de leer
- Inmediatamente se habilita el bus de datos por donde sale el contenido de esa posición de memoria sin destruirse la información que estaba almacenada.

A todo este tiempo se lo denomina "ciclo de memoria" que utiliza varios ciclos de máquina para completar la operación (sea de lectura o de escritura).

Ciclo de almacenamiento: Es el tiempo necesario para transferir un carácter (o byte) desde o hacia la memoria central.

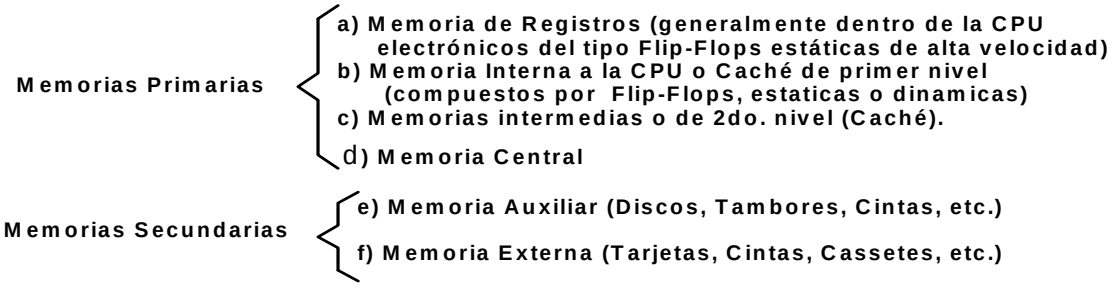
5.A.1.2. JERARQUÍAS DE ALMACENAMIENTOS.

Para seleccionar las memorias a instalar en un sistema de computación, se debe considerar los siguientes factores:

a) Capacidad de almacenamiento.
b) Tiempo de acceso.
c) Velocidad de transferencia.
d) Densidad de almacenamiento.
e) Tipo de memoria y tecnología
f) Precio.

Este último punto (precio o costo) determina los diferentes niveles o jerarquías del almacenamiento en un sistema de cómputo y resulta directamente de la velocidad (a menor tiempo de acceso más caras).

Dentro del sistema se distinguen los siguientes niveles de memorias:



Las memorias de registros: son el almacenamiento fundamental que utiliza la CPU para la operación de la máquina. Es una memoria de semiconductores muy rápida, cuyo tiempo de acceso es muy pequeño. Los acumuladores, flags, registros y contadores, entran en esta categoría.

La memoria interna: es aquélla construida internamente dentro del chip de la CPU, en que se almacenan todas las instrucciones del programa usuario y los datos que están bajo proceso inmediato, además de ciertas instrucciones pertenecientes al Sistema Operativo; a esta memoria tiene acceso la CPU y también se la conoce con el nombre de Memoria Caché de primer nivel. En el caso del procesador P6 Se integra un banco de memoria caché de 256 KBy o mas dentro del mismo Chip. En ese caso esta memoria es de segundo nivel.

La Memoria Caché es una memoria intermedia externa al procesador, generalmente se la designa como de segundo nivel, entre la CPU y la memoria Central. Es de alta velocidad de acceso, utilizada básicamente para disponer de las instrucciones y datos cerca de la CPU cuando las necesite.

Memoria Central es la que recibe los programas y los datos para su ejecución. Solamente se puede ejecutar si está cargado en ésta memoria. Generalmente está compuesta por varios MeBy y es algo más lenta que la caché. Es una memoria de Lectura y escritura que se llama RAM y una parte es solamente de lectura (ROM). Será tratada a continuación.

La memoria auxiliar o secundaria: es aquella memoria utilizada durante el procesamiento en la que se almacenan todos aquellos programas, subrutinas, datos, etc. que no han podido ser almacenados en la memoria interna debido a las limitaciones de capacidad. A pesar de que su tiempo de acceso es superior a las memorias internas, permite almacenar una gran cantidad de bits en forma muy económica. Los discos, cintas, tambores, cassettes, etc. pertenecen a este nivel.

Memoria externa: es aquélla que permite almacenar cantidades ilimitadas de información a muy bajo costo en el caso de que se requiera un bajo tiempo de acceso (caso de las tarjetas o cintas o discos ópticos).

5.A.1.3. Características de las memorias:

a) ***Volatilidad***: es la facilidad con que las memorias pueden perder la información almacenada cuando se interrumpe la alimentación (Ej.: los Flip-Flops de semiconductores). Los materiales magnéticos no son volátiles (Ej.: discos).

b) ***Forma de acceso***:

1- Acceso aleatorio o RANDOM o directo: cualquier información contenida en la memoria y cuya dirección sea conocida, puede ser alcanzada en un determinado momento. (Ej. las matrices de Flip-Flops)

2- Acceso secuencial: la información aparece en forma de una secuencia, de manera que para acceder a una determinada posición, debe pasarse previamente una serie de ellas, no requeridas (Ej. cintas magnéticas).

c) **Destructivas de la información (en la lectura):**

1- Destructiva: la información almacenada se pierde cuando es leída y se debe volver a re-escribir sobre ella para mantenerla. (Ejemplo: núcleos magnéticos).

2- No destructiva: la información no se pierde cuando es leída. (Ejemplo: los Flip-Flops).

5.A.1.4. Organización de la memoria interna (central o principal):

La organización se efectúa lógicamente en palabras cuya longitud puede ser de 4, 8, 12 o más bits, dependiendo del tipo de Hardware y de la arquitectura del computador.

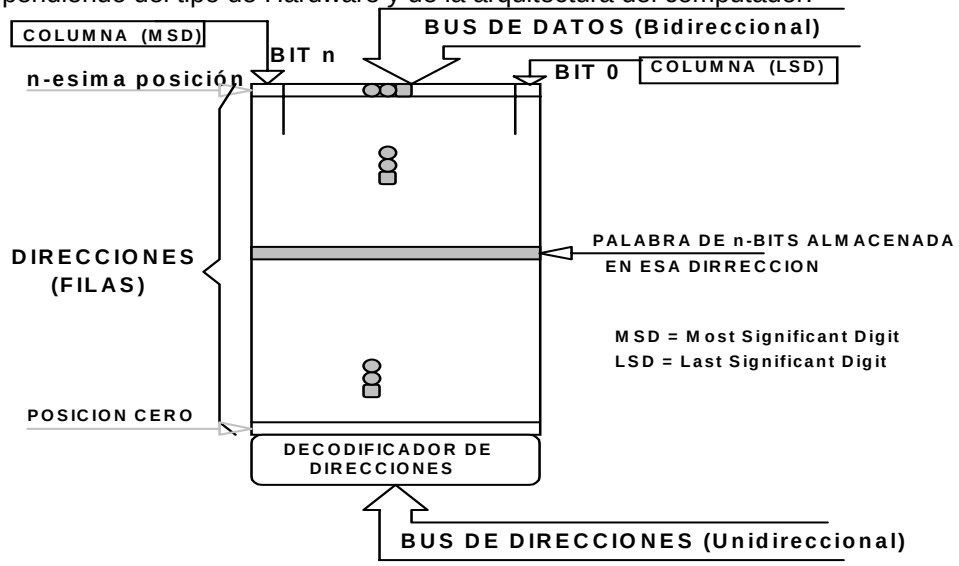


Figura 5.A.1. Elementos componentes de la memoria central

En la Fig. 5.A.1 se representa una organización típica en forma matricial. Las columnas van desde el bit 0 (posicional), hasta el bit de peso "n". Esto es el ancho de la palabra, o sea, su tamaño, también llamado ***resolución***, que debe coincidir con el ancho del bus de datos. En las filas vienen representadas las direcciones o posiciones en forma vertical, siendo la primera posición, la dirección cero, luego la segunda, la dirección uno y así hasta alcanzar la enésima que representa su tamaño o capacidad máxima, expresada en x-palabras para almacenar información.

Para referirse a una cierta palabra de la memoria, se accede mediante el bus de direcciones, quien va conectado a un descodificador, que selecciona en última instancia, la posición física de la fila en que está ubicada dicha palabra.

En la figura no se incorporó los elementos que actúan como control. Ejemplo las señales de lectura - escritura, de selección de circuitos, reset, registros separadores de bus, etc.

5.A.1.5. PROTECCIÓN DE PARIDAD

Generalmente cada posición de memoria central tiene asociada un bit adicional, inaccesible para el usuario, a efectos de poder efectuar un control de paridad. El estado de este bit se calcula automáticamente después de cada operación y se compara con el que tenía antes de realizar la misma a modo de verificación. Un ejemplo de este tipo de memorias son las EDO.

5.A.1.6. ORGANIZACIÓN DE LA INFORMACIÓN EN LA MEMORIA

La información puede organizarse, en la memoria de un computador, según distintos conceptos: los más comunes son:

Vector: es la organización más clásica de las informaciones en memoria. Consiste en almacenarlas sucesivamente, una detrás de otra. Generalmente ésta es el caso de las instrucciones de un programa, un conjunto de números diferentes, etc. Se necesita entonces, para buscar las informaciones sucesivas, de un "***pointer" o puntero*** cuyo valor sea igual a las direcciones de las informaciones almacenadas en secuencia. Este puntero generalmente es el contador de Programa (P.C.) cuyo contenido apunta a la dirección de memoria. Si son datos el ***puntero a los datos (data pointer)*** y si es una pila el stack pointer.

Lista: en la estructura de la lista cada dato va acompañado por un "pointer" que contiene la dirección del dato siguiente. Todas las operaciones se realizan entonces, por manipulación de "pointers".

Tabla: en este tipo de organización se establece una correspondencia entre dos tipos de información: la información propiamente dicha y la información de entrada que permite localizar la misma. En el caso general, la tabla se almacena bajo la forma de un vector, cada una de sus elementos contendrá la información de entrada o identificadora y la información significativa buscada. La búsqueda en tabla exige sucesivas comparaciones de la información de entrada con todas las posibles informaciones de entrada, hasta llegar a la coincidencia. En ese caso, ya está disponible la información propiamente dicha. Un ejemplo de este tipo de estructura es usada en las memorias caché.

Cola: las informaciones agrupadas según el concepto de cola funcionan según el principio **P**rimero en **E**ntrar **P**rimero en **S**alir (**PEPS** y en inglés **FIFO**: **F**irst **I**n **F**irst **O**ut). Es decir, se las ha procesado en el mismo orden en el cual ingresaron.

Pila: la información agrupada según un pila (STACK) funciona según el principio: último en entrar, primero en salir (**LIFO: L**ast **I**n **F**irst **O**ut). Es decir, la información se procesa en orden inverso al cual ingresaron.

5.A.1.7. Jerarquías de memorias de un computador:

Las razones para establecer una jerarquía en las memorias dentro de un computador reside fundamentalmente en tres factores: tamaño, tiempo de acceso y relación precio-desempeño. De estos tres, el fundamental es el tiempo de acceso que determina la velocidad en el procesamiento de los datos almacenados por consiguiente el costo del bit almacenado.

Generalmente, los computadores modernos implementan un sistema de Memoria Virtual consistente en una estructura jerarquizada, de por lo menos dos niveles, que es administrado por el Sistema Operativo, de manera que aparece ante los usuarios como una gran memoria central directamente direccionable.

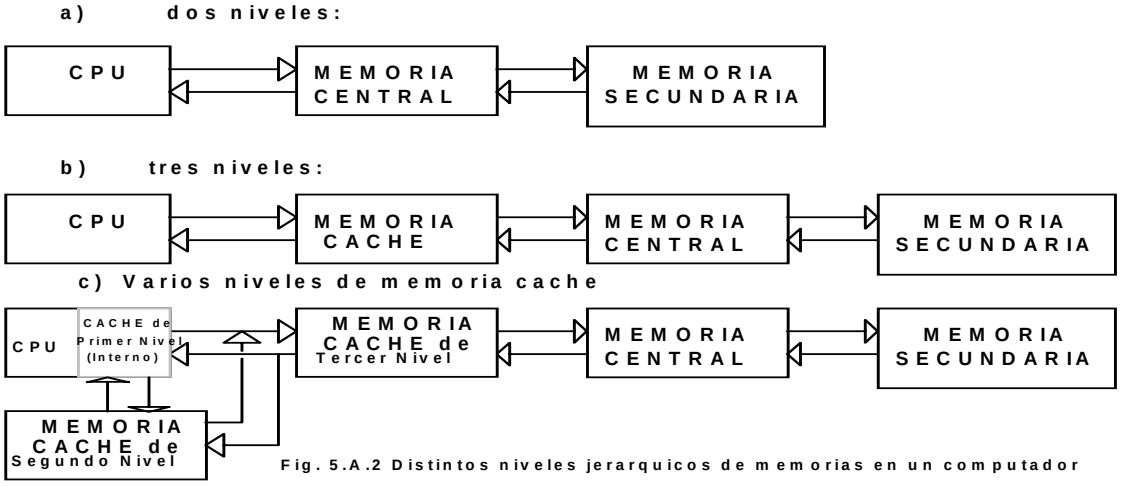


Fig. 5.A.2 Distintos niveles jerarquicos de memorias en un computador

Las jerarquía mas comunes son las indicadas en la figura 5.A.2.

Existen tres razones principales para emplear memorias virtuales:

1. Libera a los usuarios de la necesidad de ocuparse de la administración de la memoria.
2. Hace que los programas sean independientes de la configuración y capacidad de los sistemas de memoria empleados durante su ejecución.
3. Permite compartir eficientemente el espacio de memoria entre usuarios diferentes, y permite frecuencias de acceso elevadas con un costo por bit bajo, según el tipo de jerarquía empleada

La mayoría de los sistemas de memoria virtual emplean una jerarquía de dos niveles (M_1, la memoria central, de capacidad S_1 relativamente pequeña, y M_2, normalmente discos, de mucho mayor capacidad S_2).

El usuario corriente que programa en lenguaje de alto nivel, ve el sistema como una única memoria virtual de capacidad acotada por S_1+S_2. El sistema provee mecanismos automáticos para convertir las direcciones lógicas del programa del usuario (dir) en direcciones de memoria físicas P entonces distinguimos dos elementos básicos:

1. Una función de traducción de direcciones (f : dir $\Rightarrow$ P)
2. Una jerarquía de memoria.

Los distintos elementos de un sistema de memoria típico pueden pensarse como constituyentes de una jerarquía de memorias: M_1, M_2, M_3, ..., M_n, donde cada elemento Mi está en cierta forma subordinado al precedente de la jerarquía M_i - 1.

La CPU (y otros procesadores si los hubiera) se comunican directamente con el primer miembro de la jerarquía M_1; M_1 se comunica con M_2, y así sucesivamente, a través de los mecanismos de referencia y extensiones.

Si llamamos:

C_i al costo por bit de la memoria M_i, y
T_{ai} al tiempo de acceso al nivel $_i$ y
S_i a la Capacidad de almacenamiento del nivel $_i$,

las siguientes relaciones se verifican normalmente entre los miembros de la jerarquía :

$$C_i > C_i + 1;$$
$$T_{ai} < T_{ai} + 1;$$
$$S_i < S_i + 1.$$

El objetivo de diseño de una jerarquía de memoria es obtener un desempeño próximo a la del dispositivo más rápido (M_1) con un costo por bit cercano al más barato (M_n).

El desempeño de la jerarquía depende de una variedad de factores que interactúan en forma compleja. Los más importantes son los siguientes:

1. La estadística de referencias a memoria, es decir, el orden y frecuencia con que los programas generan direcciones lógicas que usan la jerarquía.
2. El tiempo de acceso T_{ai} de cada nivel M_i con respecto a la CPU.
3. La capacidad de almacenamiento de cada nivel.
4. El tamaño de los bloques de información transferidos entre representantes sucesivos de la jerarquía.
5. Los algoritmos de asignación de memoria que determinan a que áreas de memoria se asignan los bloques de información transferidos desde otros niveles.

Todos estos factores interactúan en forma que aún no está del todo comprendido. Los estudios realizados, mediante métodos de simulación, revelan solamente las relaciones generales entre estos factores. Las técnicas de simulación, son herramientas fundamentales para el diseño y evaluación de sistemas de memorias.

Para simplificar la exposición, nos restringiremos en adelante a tratar las jerarquías de dos niveles: M_1 y M_2. Las consideraciones sobre costo y desempeño pueden fácilmente generalizarse a una jerarquía de "n" niveles.

El costo promedio por bit de memoria está dado por:

$$\mathbf{C} = \frac{\mathbf{C_1 * S_1 + C_2 * S_2}}{S_1 + S_2} \quad (1)$$

donde $\mathbf{C_i}$ denota el costo por bit de $\mathbf{M_i}$, y $\mathbf{S_i}$ la capacidad de $\mathbf{M_i}$.

Para lograr el objetivo de hacer **c** muy próximo a $\mathbf{c_2}$, es claro que $\mathbf{S_1}$ debe ser pequeño comparado con $\mathbf{S_2}$.

Una de las medidas del desempeño de una jerarquía de dos niveles suele medirse mediante el ***cociente de aciertos*** **"H"** que se define como la probabilidad que una dirección lógica generada por la CPU se refiere a la información almacenada en $\mathbf{M_1}$. El cociente de aciertos generalmente se obtiene en forma experimental mediante la ejecución de un conjunto de programas suficientemente representativos, o mediante la simulación de la ejecución de los mismos, y se registra el número de referencias a $\mathbf{M_1}$ y $\mathbf{M_2}$ (denotados como $\mathbf{N_1}$ y $\mathbf{N_2}$ respectivamente). Entonces:

$$H = \frac{N_1}{N_1 + N_2} \quad (2)$$

Es claro que **H** depende fuertemente de las características del programa así como de los algoritmos de asignación de memoria y el tamaño de $\mathbf{M_1}$.

Si $\mathbf{T_{a1}}$ y $\mathbf{T_{a2}}$ denotan los tiempos de acceso a $\mathbf{M_1}$ y $\mathbf{M_2}$ respectivamente con respecto a la CPU, el ***tiempo promedio*** **"TPA"** para que la CPU acceda una palabra cualquiera en el sistema de memoria es:

$$\mathbf{TPA = H * T_{a1} + (1 - H) * T_{a2}} \quad (3)$$

En la mayoría de los sistemas de dos niveles, al hacerse una referencia a una palabra que no está en $\mathbf{M_1}$, un bloque completo de información es transferido a $\mathbf{M_1}$ desde $\mathbf{M_2}$. Solo cuando ésta

transferencia ha sido completada, la palabra buscada es accedida en memoria central. ***El tiempo de transferencia o reemplazo de bloques*** **"TB**" es tal que:

$$T_{a2} = T_{a1} + (1 - H) * TB \quad (4)$$

Como la transferencia de bloques usualmente involucra operaciones de entrda-salida lentas, entonces **TB** es muy grande comparado con T_{a1}, y similar a T_{a2}.

Si llamamos "**r**" a la relación del cociente entre los tiempos de accesos a dos niveles de memoria:

$$r = \frac{T_{a2}}{T_{a1}} \quad (5)$$

$$e = \frac{T_{a1}}{TPA} \quad (6)$$

Podemos indicar al factor "**e**" como indicador de la diferencia del tiempo de acceso promedio respecto al primer dispositivo de la jerarquía, de la ecuación (3) obtenemos la eficiencia de los procesos como:

$$e = \frac{1}{r + (1 - r) * H} \quad (7)$$

$$e = \frac{T_{a1}}{TPA} \quad (8)$$

La figura 5.A.3 gráfica a "**e**" como función de "H" y muestra la importancia de obtener elevados valores de "H" para que "e" sea próximo a uno, es decir, TPA similar a T_{5A}.

Otro indicador de la eficiencia de un sistema se puede obtener mediante el ***factor de utilización*** "**u**" de la memoria. La eficiencia con que la memoria está siendo utilizada en un instante dado puede determinarse como:

$$u = \frac{S_u}{S} \quad (9)$$

donde S_u es el tamaño de la memoria ocupada con partes activas de programa del usuario, y **S** es la capacidad total de ese nivel. Dado que la memoria central es mucho más valiosa que la memoria secundaria, "**u**" es útil para medir el empleo de la memoria central. Las zonas que son comprendidas por ($S - S_u$) son las que se desperdician debido al fenómeno llamado de ***fragmentación externa o fragmentación interna***, o que están ocupadas por el Sistema Operativo, o áreas de datos del sistema administrador de memoria, etc., o las ocupadas por los programas de usuarios que no están ejecutándose activamente

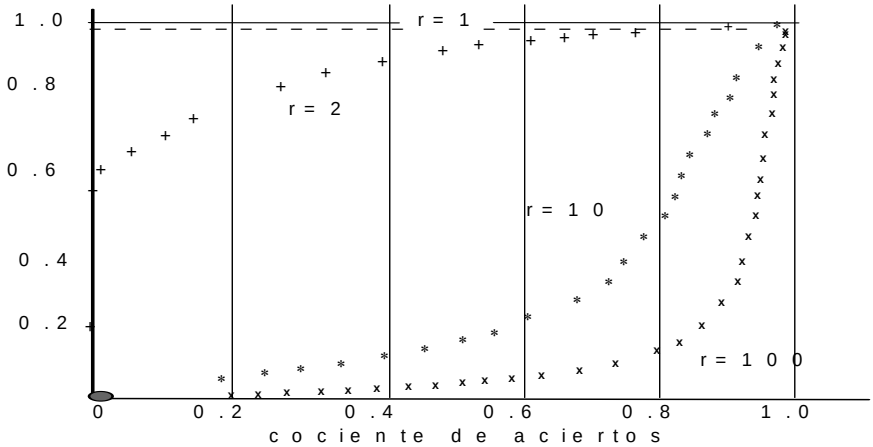


Fig. 5.A.3 Gráfico de cociente de aciertos

AUTOEVALUACIÓN DEL MODULO 5:

Preguntas:

1.- Que función cumple la TLB?

2.- Cuál es el dispositivo de HW encargado de la traducción de las direcciones lógicas de memoria a direcciones físicas?

3.- ¿Cómo se protege el área del S.O. en una asignación de memoria contigua simple o monitor residente?.

4.- ¿Qué diferencia hay entre las fragmentaciones internas y externas de memoria?, ¿De qué formas se pueden solucionar (o atenuar) estos problemas?

5.- ¿Qué ventajas tienen la segmentación y paginación en memoria virtual en comparación con los modelos sin memoria virtual?

6.- ¿Qué diferencias hay entre los algoritmos LRU y FIFO?

7.- ¿Como es el funcionamiento del algoritmo de reemplazo optimo (OPT o MIN) ?

8.- En el algoritmo de paginación por demanda con TLB. Empieza la rutina de tratamiento de la Page Fault, la condición de "memoria llena" ¿que alternativas posee ?

9.- ¿Qué es "trashing"?. Brevemente explique el principio de localidad..

10.- ¿Qué es el Translation Lookaside Buffer?

11.- ¿Como se combinan la paginación y la segmentación?

12.- ¿Qué es Page buffering?

13.- ¿Qué es Swapping?

14.- Una decisión importante dentro del diseño de hardware es el tamaño de las páginas. Enumere algunos factores a considerar

15.- ¿Por qué no es posible forzar la protección de memoria en tiempo de ejecución?

Múltiple Choice:

1.- ¿Cuál de las siguientes características pertenece a paginación por demanda?: a) Deja el espacio libre en memoria en un solo bloque. Logrando así el máximo aprovechamiento de esta. b) Aumenta el nivel de multiprogramación. c) Genera perdida de tiempo de CPU (Overhead) d) Conviene aplicar la compactación cuando un proceso no puede ejecutarse por falta de memoria. e) Todas las anteriores son correctas. f) Todas las anteriores son incorrectas	**2.- ¿Cuáles de los siguientes son los objetivos de la administración de memoria central?** a) Tener siempre actualizados los PCB. b) Mecanismos de protección de memoria, áreas del sistema y áreas de usuario en sus distintos niveles. c) Asignar, liberar y reasignar memoria. d) Mantiene un control de las direcciones libres, los ocupados y los defectuosos. e) Uso compartidos de Códigos y datos f) Generación de Direcciones físicas y direcciones lógicas g) Ninguna de las anteriores es correcta.
3.- La fragmentación externa es: a) Es la generación de un gran número de áreas libres de gran tamaño b) Es la generación de un gran número de áreas libres de tamaño pequeño c) Es el desaprovechamiento de un trozo de memoria dentro de una página d) Es el desaprovechamiento de un trozo de memoria dentro de un frame. e) Todas las anteriores son correctas. f) Todas las anteriores son incorrectas	**4.- En la memoria Virtual:** a. El programa se divide en un conjunto de frames de igual tamaño b. Cada proceso se divide en paginas del mismo tamaño que el frame. c. Un proceso se carga con todas las paginas en memoria en frames libres no contiguos. d. Hay fragmentación externa. e. El Hardware de direccionamiento y reubicación hace más lento al sistema. f. Todas las anteriores son correctas. g. Todas las anteriores son incorrectas
5.- **En Buddy System** a. Hardware es sencillo y de poco overhead en la funciones de liberar o asignar espacio.	**6.- La carga dinámica se caracteriza por…** a) Las rutinas se cargan sin que se las llamen. b) Las rutinas se almacenan en memoria en formato de carga

b. Es Muy rápido. c. Hace un mal aprovechamiento de la memoria por la fragmentación d. Utiliza el Swapping para sacar de memoria los segmentos inactivos y para cargar uno nuevo. e. Todas las anteriores son incorrectas f. Ninguna de las anteriores es correcta.	relocalizable c) Obtener una mejor utilización del espacio de memoria. d) Una rutina no se carga hasta que se le llame. e) Siempre se cargan las rutinas aunque no se las usa f) Todas las anteriores son incorrectas
7.- Cuál de las siguientes políticas implantados en un sistema operativo se usan para administrar eficientemente la memoria real (no virtual)…. a) Políticas de "fetch" o búsqueda para decidir cuándo debe traerse una página a memoria. b) Política “Demand paging” es la que determina que se debe cargar a memoria una página cuando ésta se necesita. c) Política “prepaging” se trata de traer una página a memoria antes de que ésta se necesite d) Política “placement" o colocación para decidir en qué dirección de memoria se cargará la página a traerse a memoria. e) Política “replacement" o reemplazo se utilizan para decidir qué página debe sacarse de memoria para hacer espacio a una nueva que llega. f) Ninguna de las anteriores se usan.	**8.- En la sustitución global de frames….** a) El proceso puede tomar un frame a sustituir de cualquier lugar de memoria central aunque el frame esté asociado a otro proceso. b) El proceso debe tomar un frame de su propio conjunto de frames asignados a él. c) El número de frames asociados al proceso no cambia. d) El proceso sólo está afectado por su propio paginado. e) Los procesos no son responsables ni pueden controlar su propio page fault rate. f) Todas las anteriores son incorrectas
9.- El Second-Chance Algorithm se caracteriza por…. a) El algoritmo tiene dos punteros, un puntero va delante del otro b) La lista de páginas candidatas a ser sustituidas se mantiene en una cola circular c) Usar una cola FIFO modificado con un reference bit. d) Si una página es seleccionada como víctima, primero se verifica su reference bit. e) Mantener un contador por cada página del número de veces que fue accedida. f) Sustituir la página con el tiempo de permanencia máximo. g) Todas las anteriores son correctas. h) Todas las anteriores son incorrectas	10.- **Entre las desventajas de LRU se destacan…** a) La anomalía de Belady. b) La implantación de este algoritmo no podría concebirse sin la ayuda del hardware. c) La actualización de los registros de reloj o pila debe efectuarse para cada referencia a memoria. d) Mediante interrupciones para cada referencia lo hace unos diez veces más lento e) Que genera un excesivo trashing. f) Todas las anteriores son correctas. g) Todas las anteriores son incorrectas
11.- ¿En qué instante se toma la decisión de compactar la memoria? a) Inmediatamente después que se desocupa una partición. b) Sólo compactar cuando es necesario c) Cuando aparece un pedido de asignación que no pueda ser satisfecho d) Cuando la compactación es posible e) Todas las anteriores son correctas. f) Todas las anteriores son incorrectas	**12.- En la administración de Memoria segmentada. Un segmento se caracteriza por….** a) Ser un conjunto lógicamente relacionado de datos o códigos, generado por el compilador. b) Ser un conjunto lógicamente relacionado de datos o códigos, generado por el usuario. c) Ser su tamaño igual y fijo definido por el compilador d) Tener cada segmento asociado siempre un área fija de memoria y un tamaño. e) Ser su tamaño arbitrario y definido por el usuario según el propósito del segmento en el programa. f) Tener cada segmento asociado un nombre o número y una longitud. g) Todas las anteriores son correctas. h) Todas las anteriores son incorrectas
13. En la administración de Memoria paginada. Compartir una página se caracteriza por…. a) Ser utilizado en un entorno de tiempo compartido. b) No ser utilizado en un entorno de tiempo compartido. c) Ser una forma de compartir la memoria entre procesos similar al de compartir hilos d) Ahorrar considerablemente el espacio de memoria. e) Ser el S.O. quien deberá asegurar la propiedad de sólo lectura del código compartido f) Todas las anteriores son correctas. g) Todas las anteriores son incorrectas	**14.- En la administración de Memoria paginada. Las principales ventajas son….** a) Produce Fragmentación interna. b) El Hardware es mas caro. c) Elimina el problema de fragmentación externa. d) Menor velocidad debido a la constante traducción de direcciones lógicas en físicas. e) No necesita compactación. f) Gasta más memoria central utilizado con tablas. g) Mayor grado de multiprogramación. h) Todas las anteriores son correctas. i) Todas las anteriores son incorrectas
15.- En la Administración de Memoria con particiones variables. Las principales ventajas son…. a) Produce Fragmentación interna. b) El Hardware es poco costoso. c) Elimina el problema de fragmentación externa. d) No necesita compactación. e) Los algoritmos son simples y fáciles de implementar. f) Mayor grado de multiprogramación. g) Todas las anteriores son correctas. h) Todas las anteriores son incorrectas	**16.- En la administración de Memoria con particiones variables. La técnica de asignación…** a) Best fit es lento y deja particiones libres pequeñas b) Worst fit usa huecos mas grandes y permite mejor aprovechamiento de la memoria c) El First Fit es muy lento pero es el mas usado d) El Next Fit es muy eficiente porque no produce fragmentación externa e) El swapping se utiliza siempre f) Todas las anteriores son correctas. g) Todas las anteriores son incorrectas
17.- **En la administración de Memoria con segmentación con paginación por demanda. Se caracteriza por …**	**18.- En la administración de Memoria, la protección …** a) Debe ser realizada por el M.M.U. (Hardware) b) Debe ser realizada exclusivamente por el S.O.

a) Combinar las técnicas de Segmentación y de paginación. b) En general los segmentos tienen un tamaño múltiplo de páginas. c) No se necesitan tener cargadas en memoria central todas las Páginas de los segmentos. d) La dirección virtual se organiza en tres partes segmento, pagina y desplazamiento e) Todas las anteriores son correctas. f) Todas las anteriores son incorrectas	c) Tiene que controlar todos los accesos de los threads a memoria. d) Tiene que controlar todos los accesos de los procesos y del S.O. a memoria e) Todas las anteriores son correctas. f) Todas las anteriores son incorrectas
19.- El Conjuntos de trabajo (Working Set) es… a) Un programa generalmente está compuesto por varias localidades distintas b) El conjunto de páginas cargadas en frames en la memoria central. c) La Localidad o conjunto de páginas usadas por un proceso en un dado momento. d) La demanda superior al tamaño de la memoria central o sea thrashing e) Una técnica que trae páginas por adelantado para evitar un gran número de page faults. f) Todas las anteriores son correctas. g) Todas las anteriores son incorrectas	20.- **La Cadena de distancias es...** a) La cadena formada por las respectivas distancias de recuperación de las páginas que son solicitadas para la ejecución de una tarea b) La cadena formada por las referencias de las páginas que son solicitadas para la ejecución de una tarea c) Aquella que tiene la misma cantidad de elementos que la cadena de referencias. d) La cadena formada por las referencias de las páginas que son enviadas al disco en el intercambio durante la ejecución de una tarea e) Todas las anteriores son correctas. f) Todas las anteriores son incorrectas

Respuestas a las preguntas

1.- ¿Que función cumple la TLB?

El TLB cumple la función de guardar la traducción de direcciones virtuales o lógicas a físicas de las últimas paginas que se han usado. Trabaja como una memoria caché ultrarrápida.

2.- ¿Cuál es el dispositivo de HW encargado de la traducción de las direcciones lógicas de memoria a direcciones físicas?

El MMU es la encargada de dicha traducción. Los programas nunca ven direcciones físicas reales, sino que siempre trabajan con direcciones lógicas.

3.- ¿Cómo se protege el área del S.O. en una asignación de memoria contigua simple o monitor residente?.

Se protege mediante un registro llamado registro frontera (fence register y un modo dual de trabajar la CPU supervisor / usuario).

4.- ¿Qué diferencia hay entre las fragmentaciones internas y externas de memoria?, ¿De qué formas se pueden solucionar (o atenuar) estos problemas?

La **fragmentación interna** ocurre cuando un programa es asignado a un bloque de memoria entero muy grande para lo que necesita, desperdiciándose de esta manera una gran cantidad de memoria.
La **fragmentación externa** ocurre cuando van quedando espacios no asignados de memoria entre otros asignados. Los espacios no asignados usualmente van quedando muy chicos y no alcanzan para poder ser asignados a programas, pero si se suman todos a lo largo todo el espacio de memoria, se ve que el desperdicio de la misma también es grande.
La fragmentación interna se puede solucionar utilizando particiones dinámicas que asignen solo el espacio necesario (no se suele utilizar porque tiene otras desventajas) o segmentación; está atenuada en la paginación, pero igual puede ser grave en ciertos casos. La fragmentación externa se puede solucionar con el uso de compactación a periodos determinados (no es muy aconsejable) o con paginación.

5.- ¿Qué ventajas tienen la segmentación y paginación en memoria virtual en comparación con los modelos sin memoria virtual?

Las ventajas de estos dos modelos con memoria virtual sobre sus semejantes sin ella son que ahora no es necesario tener a todo el programa en memoria central, ya que solo se cargan las páginas necesarias y se hacen swaps entre memoria central y la virtual. Esto además facilita la multiprogramación, ya que hay más espacio en memoria central para varios programas. Otra ventaja es la longitud de los programas, que puede ser mucho mayor, al estar almacenados en forma auxiliar hasta ser swapeadas a memoria central las páginas necesarias para su ejecución.

6.- ¿Qué diferencias hay entre los algoritmos LRU y FIFO?

El algoritmo LRU es el más cercano a lo ideal, en términos puramente de corrección de operación, ya que privilegia la estancia de las páginas más usadas en memoria, cosa que FIFO no tiene en cuenta, al sacar las más antiguas. En contrapartida, LRU requiere de más complicaciones en su implementación y además produce overhead y es in aplicable en memorias grandes, a diferencia de FIFO que es más simple y no produce overhead. Por este dilema se desarrollaron algoritmos tales como los que utilizan clock y los de page buffering.

7.- ¿Como es el funcionamiento del algoritmo de reemplazo optimo (OPT o MIN) ?

A cada momento i, se calcula el tiempo Tj que falta para que se efectúe una referencia a la pagina Pj.
Posteriormente se sustituye la pagina para la cual Tj es máximo.

8.- En el algoritmo de paginación por demanda con TLB. Empieza la rutina de tratamiento de la Page Fault, la condición de "memoria llena" ¿que alternativas posee ?

Si es memoria llena se ejecuta el reemplazo de pagina según la estrategia de reemplazos y posteriormente el S.O. Ejecuta la lectura de paginas del disco y activa el hardware de E/S. En cambio si no es memoria llena directamente Ejecuta la lectura de paginas del disco.

9.- ¿Qué es "trashing"?. Brevemente explique el principio de localidad..

El trashing se produce cuando el procesador se pasa mas tiempo intercambiando páginas de procesos con la E/S que ejecutando instrucciones.
El principio de localidad establece que las referencias al programa y los datos en un proceso tienden a agruparse. Se asume que en un corto período de tiempo, solo se necesitarán unas partes del proceso, es posible predecir cuáles páginas se necesitarán. Esto evitará el trashing.

10.- ¿Qué es el Translation Lookaside Buffer?

Cada referencia a memoria virtual puede causar dos accesos a memoria física: uno para la búsqueda en la tabla de páginas, y otro para la búsqueda de los datos. De esta manera, el tiempo de acceso a la memoria seria el doble. Para solucionar dicho exceso de tiempo, la mayoría de los administradores de memorias usan una memoria asociativa llamada Translation Look-aside Buffer, que es una memoria caché especial en registros de alta velocidad para tablas de paginas dentro del M.M.U.. Tiene las mismas funciones que una caché y contiene aquellas tablas de paginas que fueron usadas mas recientemente.

11.- ¿Como se combinan la paginación y la segmentación?
La paginación es transparente al programador, y no produce fragmentación externa pero sí interna, y la segmentación es visible al programador y permite crecer a las estructuras de datos, modularizar, tener un soporte para la protección. Para combinar las ventajas de ambas algunos sistemas están equipados con procesadores en que el hardware y sistemas operativos soportan tanto paginación como segmentación.
El espacio de direccionamiento se parte en un número de segmentos, los que a su vez se dividen en un número de páginas de igual tamaño. Esto para el programador sigue siendo una dirección lógica con sus dos partes (numero de segmento y desplazamiento).

12.- ¿Qué es Page buffering?
Es una estrategia que aumenta la performance de la paginación y permite un simple sistema de reemplazo de páginas. La forma de reemplazo es similar a la FIFO. La página reemplazada se agrega a una de estas dos listas, lista de Páginas libres (si la página no ha sido modificada), o lista de páginas modificadas.

13.- ¿Qué es Swapping?
Swapping es el intercambio de información entre dos niveles de memorias. En este caso Memoria Central y Memoria Secundaria basado en un dispositivo de acceso directo (DASD: Direct Access Storage Device).
A la operación de enviar información al disco se le llama swap out, y al de cargar la información desde disco a memoria se le llama swap in. Se pueden añadir intercambios a cualquier algoritmo. A intervalos determinados por el S.O., generalmente indicados por las politicas de planificación de la CPU, los procesos se copian de la memoria central al almacenamiento auxiliar y más tarde se copian de nuevo a la memoria central. Este esquema permite ejecutar mas procesos de los que caben a la vez en memoria .
Normalmente un proceso que sale de un intercambio regresa al mismo espacio de memoria que antes ocupaba. Esta restricción la determina el método de enlace de direcciones. Si el enlace se lleva a cabo durante el ensamble o carga, el proceso no puede moverse a otras localidades. En cambio, si el enlace se realiza durante la ejecución, se puede intercambiar un proceso colocándolo en un espacio de memoria distinto.
Existen distintas variables de esta técnica de almacenamiento:
- Sistema monoprogramado.
- Multiprogramado con particiones fijas.
- Multiprogramado con particiones variables.
- Sistemas de intercambio de almacenamiento virtual.

14.- Una decisión importante dentro del diseño de hardware es el tamaño de las páginas. Enumere algunos factores a considerar
Uno de los factores a considerar es la fragmentación interna. Cuanto más pequeño sea el tamaño de la página, menor es la cantidad de fragmentaciones internas y para optimizar el rendimiento de la memoria central, lo que se quiere es reducir el número de fragmentaciones internas. Por otro lado, cuanto más pequeñas sean las páginas, más paginas son requeridas por proceso y más páginas por proceso significa tablas de páginas más grandes. Para programas que se encuentran en un ambiente multiprogramación, esto significa que alguna parte de las tablas de páginas de procesos activos deban estar en memoria virtual. Entonces, habrá una doble falta de página para una sola referencia a memoria: primero para traer en la parte de la tabla de página necesaria y segundo para traer la página del proceso.
Otro factor es que las características físicas de la mayoría de los dispositivos de memoria secundarios, los cuales son rotacionales, favoreciendo a las páginas de gran tamaño a la transferencia eficiente de datos por bloques.

15.- ¿Por qué no es posible forzar la protección de memoria en tiempo de ejecución?
Los requerimientos de protección de memoria deben ser satisfechos por el procesador (hardware), más que por el sistema operativo (software). Esto es porque el sistema operativo no puede anticipar todas las referencias a memoria que un proceso realizará. Incluso si tal anticipación fuera posible, sería mucho el tiempo que consumiría en resguardar cada programa por adelantado de posibles violaciones de referencia de memoria. Por eso, es solo posible estimar la correcta referencia a memoria (acceso a datos o saltos) en el momento en que se ejecuta la instrucción que hace la referencia. Para cumplir esto el hardware del procesador debe tener tal capacidad.

Respuestas del múltiple choice.

1.- b, c.	2.- b, e, f.	3.- b.	4.- b, e.	5.- a, b, c.
6.- c, d.	7.- f.	8.- a, e.	9.- c, d.	10.- b, c, d.
11.- e.	12.- b, e, f.	13.- a, c	14.- c, e, g.	15.- b, e, f.
16.- a.	17.- e.	18.- a, d.	19.- b.	20.- a, c.