CAPITULO 13

ADMINISTRACIÓN DE MEMORIA

13.1 - INTRODUCCIÓN

Es bien sabido que tanto el Procesador como los dispositivos de E/S interactúan con la Memoria. Así también un programa debe estar en Memoria para poder ser ejecutado ya que el procesador levanta de allí las instrucciones que debe ejecutar.

Ahora bien si se quiere compartir la memoria entre varios programas, por ejemplo en un entorno de multiprogramación, y permitir además el correcto funcionamiento de los canales de E/S, se hace necesario administrar la memoria entre tales programas.

De esto último se deduce que el Administrador de Memoria es activado toda vez que se carga un trabajo. Luego veremos que no solo se activa al cargar un trabajo sino que también tiene otras funciones.

Los métodos de administración de memoria han ido evolucionando, lógicamente desde el más sencillo creciendo en complejidad; seguiremos pues ese orden.

Para todos los casos estudiados, salvo que se indique lo contrario, se supondrán memorias de direccionamiento continuo.

13.2 - Administración de Memoria SIMPLE CONTIGUA

En este sistema de administración la memoria aparece al programa como una única extensión contigua de direcciones, compartida solo por él y por el sistema operativo.

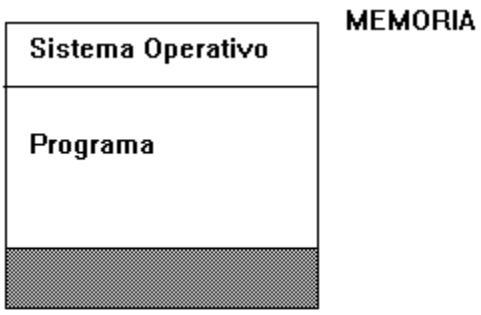


Fig. 13.1. - Administración de Memoria Simple Contigua.

Se hace necesario lograr en este tipo de administración algún mecanismo de protección para el sistema operativo que es residente.

Existen dos alternativas:

1) la utilización de un límite dado por un registro fijo por hardware o registro de protección cargado por el sistema operativo que sirve como punto de comienzo de carga del programa como en el Fortran Monitoring System IBM 3094 o como límite como en el caso de la H.P. 2116B que carga desde memoria baja. Su desventaja es que requiere la recompilación de los programas al modificarse el valor de tal registro.

2) otra opción consiste en un registro Límite o registro Base (o de Reubicación) (tal el caso de la CDC 6000).

Mediante alguno de estos dos sistemas el sistema operativo intercepta aquellas direcciones inválidas generándose entonces una interrupción por error de direccionamiento.

Por tanto esta administración requiere desde el punto de vista del **hardware** los elementos para proteger al sistema operativo, y desde el punto de vista del **software** las facilidades al programa (E/S) y la rutina de atención de interrupciones por invasión al sistema operativo.

Sus desventajas son :

- el desperdicio de recursos tanto de la memoria no utilizada, como también el desperdicio respecto de los periféricos no usados durante la ejecución del programa.
- la imposibilidad de ejecutar programas que requieran el uso de más memoria que la disponible, por más que el sistema posea una capacidad de direccionamiento mayor.

13.3 - Capacidad de direccionamiento vs. Capacidad de memoria

Por un lado se habla de memoria disponible y por otro de la capacidad de direccionamiento del sistema. El punto es el siguiente : si el direccionamiento de un sistema está dado por 24 bits esto implica que es posible direccionar hasta 16 Mbytes de memoria y no más.

Pero si la memoria disponible en este sistema es de 3 Mbytes, el tamaño máximo de programa que pueda residir en este sistema será de 3 Mbytes (descontando el espacio correspondiente al Sistema Operativo) y **no** más, aunque la capacidad de direccionamiento sea de 16 Mbytes.

Este tipo de inconvenientes se encontrará en todo sistema de Administración de Memoria que requiera que la totalidad del programa resida en memoria para poder ser ejecutado.

13.4 - Soluciones a la monoprogramación

Para resolver el problema de la residencia de un único programa en memoria que conlleva monoprogramación, y la limitación de memoria, tanto sea por su escasez como por su limitación de direccionamiento, se han utilizado algunos artilugios, de los cuales veremos un par.

13.4.1 - **Simulación de multiprogramación ó Swapping**

Esta técnica consiste en aprovechar los tiempos en los cuales un programa (proceso) se encuentra ocioso (debido a que un operador está pensando, como en el caso de un editor, o está realizando una operación de E/S sobre un periférico lento) para desalojarlo completamente de la memoria, sobre un periférico rápido, y cargar desde un periférico similar otro programa que estuviese pronto para su ejecución.

Obviamente esta técnica implica la necesidad de preservar buffers de memoria para recibir/emitir datos de

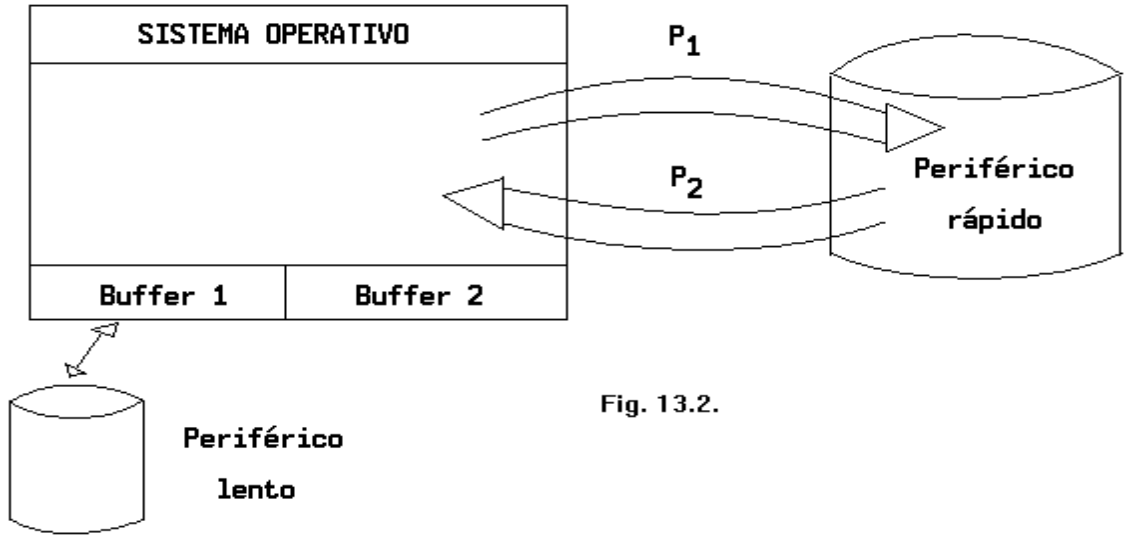


Fig. 13.2.

entrada/salida para cada proceso que conviva en un determinado momento dentro del sistema. Además la mezcla de procesos debe ser tal que ninguno sea de alto uso de procesador, pues, sino esta técnica no resultaría útil. Son necesarios además, elementos que controlen las operaciones de E/S, para permitir que el procesador pueda dedicarse a la ejecución de programas.

Esta técnica fue utilizada por el Bull 61 con cierto éxito, mientras el número de procesos fuese pequeño (4) y ninguno de alto consumo de procesador.

Actualmente esta técnica es utilizada también en sistemas operativos que manejan multiprogramación, inclusive en Arquitecturas de tipo MIMD cuando, por algoritmo, se decide castigar a un proceso por haber excedido la utilización de algún recurso (en particular el procesador, por ejemplo en la IBM OS/MVS).

13.4.2 - **Simulación de mayor direccionamiento a memoria ó Técnica de Overlay**

Esta técnica consiste en aprovechar la modularidad de los programas y en particular en la capacidad de detectar conjuntos de módulos o subrutinas que serán ejecutados en forma disjunta.

Normalmente se confecciona y vincula un programa y sus subrutinas en forma lineal, o sea dándole a cada uno de ellos su propia dirección, como por ejemplo puede verse en la Fig. 13.3, programa que no podrá ser ejecutado en una memoria de capacidad 400.

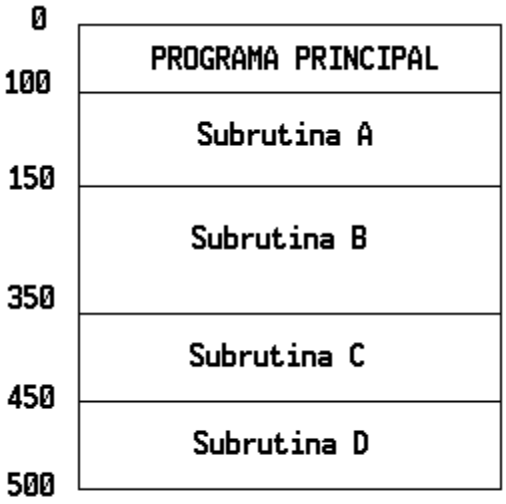


Fig. 13.3.

Ahora bien, si el programa pudiese ser descompuesto en forma lógica de la manera que se ve en la Fig. 13.4, o sea que cuando se utilicen las subrutinas SubA y SubB (rama A), no se utilizan las SubC y SubD (rama B), es decir, que el uso de la rama A es excluyente con la rama B, podría pensarse y vincularse a este programa reutilizando direcciones, o sea que podría guardarse en una biblioteca con el direccionamiento en la forma que se muestra en la Fig. 13.5 lo que permitiría que en algún momento se estuviese ejecutando la rama A

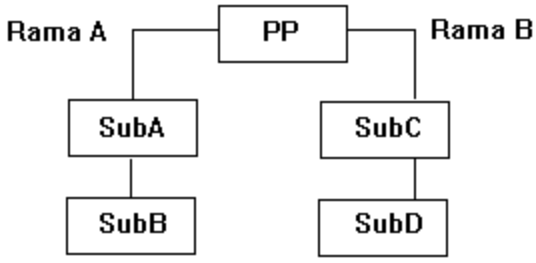
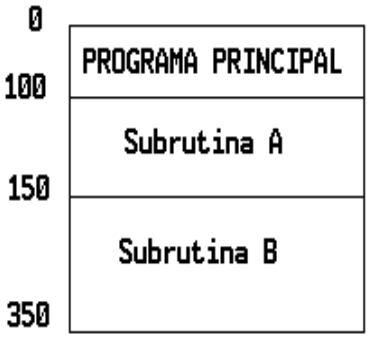
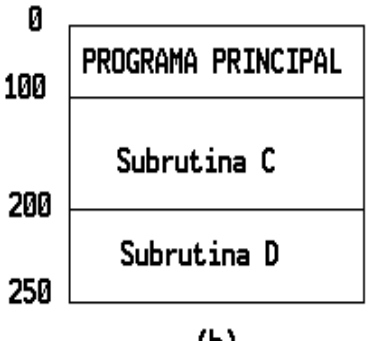


Fig. 13.4.

con un mapa de memoria como se ve en la Fig. 13.6 a) y en otro la rama B con un mapa de memoria como se ve en la Fig. 13.6 b), con lo cual ahora sí es posible ejecutar este programa en una memoria de capacidad 400.



(a)



(b)

Fig. 13.6.

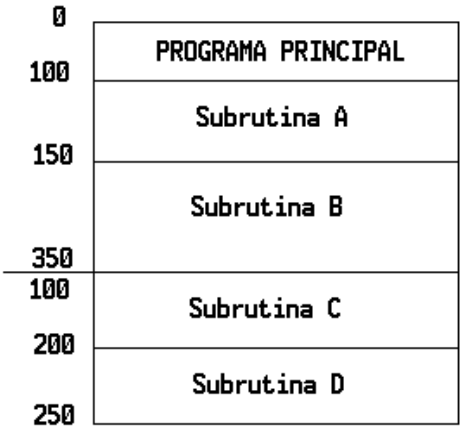


Fig. 13.5.

Esta estructura de overlay se puede pensar en forma estática, o sea previa a la carga del programa se determinará qué rama se ejecutará, y durante toda esa sesión se ejecutará esa sola; o en forma dinámica, en la cual la lógica del programa PP será quien determine qué rama se carga y si está cargada la rama A y se necesita la rama B, esta "tapará" a la rama A y será ejecutada, y así sucesivamente.

13.5 - **Administración de memoria PARTICIONADA FIJA**

En este esquema se establecen particiones fijas de la memoria de una sola vez y para siempre (por hardware o por sistema operativo), o en caso contrario esas particiones son cambiables en tamaño mientras no haya trabajos ejecutándose (normalmente esta tarea la realizará el operador desde consola).

La protección aquí se logra mediante un registro Base (también de reubicación) y un registro Límite o Longitud asociados a cada uno de los trabajos que se están ejecutando.

En este método aparece una fragmentación de la memoria debido a que parte de la partición no es utilizada por el programa.

Los mecanismos de protección y reubicación mediante los registros Base y Longitud son mecanismos **hardware** que surgen en este método de administración. Se agregan las interrupciones por direccionamiento (intento de acceder fuera de la partición).

En cuanto a **software** tenemos las rutinas de atención de interrupciones, las rutinas que se encargan de manejar las tablas de dicha administración, rutinas para asignación de archivos y dispositivos y los programas de canal.

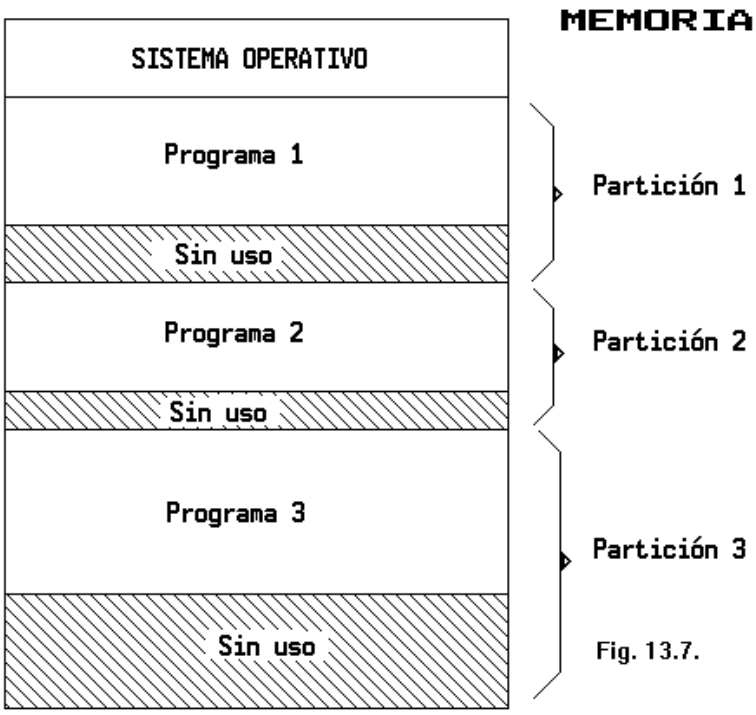


Fig. 13.7.

Las tablas necesarias para manejar este tipo de administración son las siguientes :

- Una tabla de tamaños (una sola en el sistema) que es de la siguiente forma :

PROCESO	DIRECCION	LONGITUD	ESTADO
P1			ejecutando
P2			bloqueado
P3			listo

Fig. 13.8.

- Un Bloque de Control de Proceso (BCP, uno para cada proceso que se encuentre dentro del sistema) que debe contener :

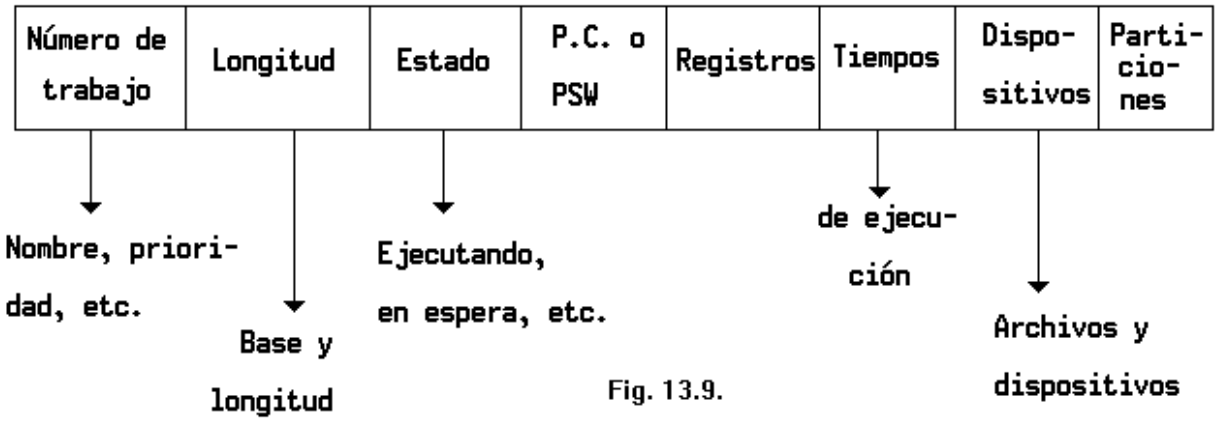


Fig. 13.9.

Obviamente este sistema de Administración de Memoria es el que comienza a permitir la ejecución en multiprogramación. Uno de los primeros existentes fue el IBM OS/360 MFT.

13.6 - **Administración PARTICIONADA VARIABLE SIN COMPACTACION.**

En este esquema las particiones se establecen según la longitud de los programas iniciales.

Al cabo de un tiempo se produce mucha fragmentación, por tanto existen diversas políticas para asignar una partición de memoria libre.

En principio sea cual fuere el mecanismo de asignación de una partición libre se hace necesario contar con la información de cuáles son esas particiones libres de memoria lo cual nos lleva a la aparición en este tipo de administración de una lista de zonas disponibles.

Existen tres grandes políticas de asignación :

1) **La primer zona libre** : aquí se recorre la tabla de espacios libres y la primer partición que se encuentre en donde quepa el programa es asignada.

2) **La mejor zona** : se recorre la tabla buscando aquella partición que mejor se ajuste al programa que se desea ingresar.

3) **La peor zona** : se recorre la tabla buscando la mayor partición libre existente, la que será asignada al nuevo proceso.

El **hardware** necesitado en esta administración coincide con el de particionada fija, en tanto que desde el punto de vista del **software** nuevo se agregan aquí las nuevas rutinas de manejo de las tablas de zonas libres para el mecanismo de asignación.

Uno de los primeros sistemas operativos que utilizó este esquema fue el IBM OS/360 MVT.

13.7 - **Administración PARTICIONADA VARIABLE CON COMPACTACION.**

Cuando la cantidad de memoria fragmentada es tal que su suma permite el ingreso de un trabajo nuevo (el cual de otra forma no podría ingresar al sistema) se "compactan" los trabajos que se encuentran en memoria.

Para realizar esta compactación es necesario contar con el registro de reubicación ya que de otra forma al mover un programa de su posición original en memoria deberían alterarse las instrucciones de tal programa.

Se puede realizar una compactación directamente sobre memoria (Memoria-a-Memoria) o utilizando dispositivos periféricos (Memoria-a-Disco-a-Memoria), en este último caso se descarga bastante trabajo de la CPU, ya que la misma puede quedar eventualmente libre para continuar la ejecución de alguno de los procesos que no se encuentran involucrados en la compactación. En el caso de la compactación Memoria-a-Memoria la CPU se encuentra dedicada a full a la tarea de compactación, por lo tanto se detienen las ejecuciones de **todos** los procesos.

Existen diversas técnicas de compactación pero ya que el mecanismo de compactación es sumamente costoso se trata de disminuir lo más posible la cantidad de veces que se compacta como así también se intenta compactar en aquellos casos en que la cantidad de bytes involucrados sea mínima.

Nuevamente aparecen aquí los mecanismos **hardware** de protección y reubicación, y se agregan como mecanismos de **software** las rutinas de compactación y reubicación y desde ya tienen las mismas tablas que indican la ubicación de los trabajos y las zonas libres.

Algunos sistemas operativos que los utilizaron fueron el GECOS del BULL 66 y el SCOPE de la CDC 6600.

13.8 - **Administración de memoria PAGINADA**.

En este esquema la memoria física se divide en Bloques de igual tamaño, cada uno de los cuales contendrá una Página de programa. Las páginas de los programas estarán ubicadas en los bloques de memoria los cuales no tienen porque ser contiguos. (Nótese esta fundamental diferencia frente a las otras Administraciones de Memoria).

Al ser cargado en proceso a memoria se crea una Tabla de Distribución de Páginas (TDP) que está apuntada desde el Bloque de Control de Proceso (BCP) y cada una de sus entradas indican en qué bloque de memoria real se encuentra cargada cada una de las páginas del programa.

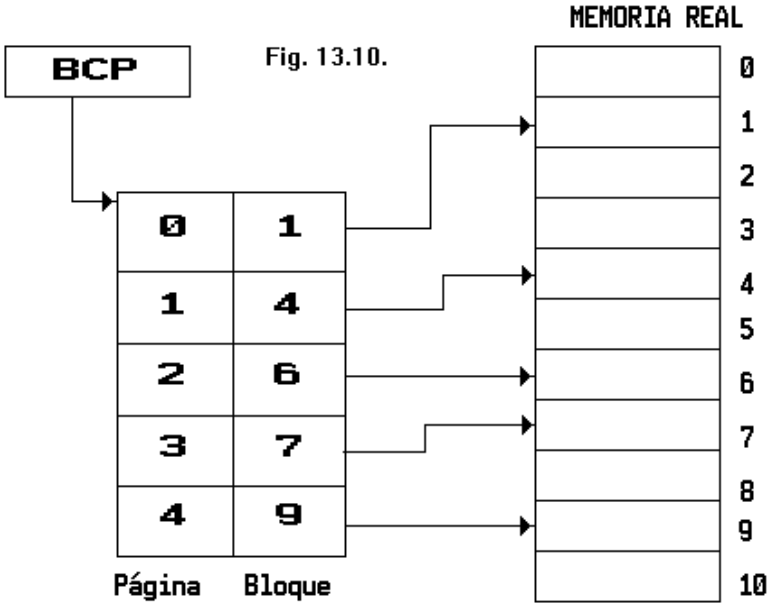


Fig. 13.10.

13.8.1 - **Cálculo de la dirección**

Las direcciones lógicas están compuestas de dos partes :

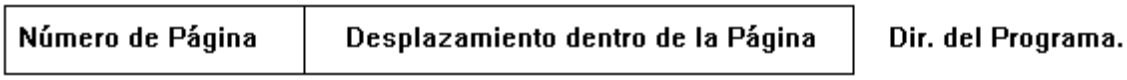


Fig. 13.11.

El número de página es tomado por el DAT (Direct Address Translator) el cual accede a la TDP y obtiene el número de bloque que corresponde a esa página para ese proceso y obtiene :

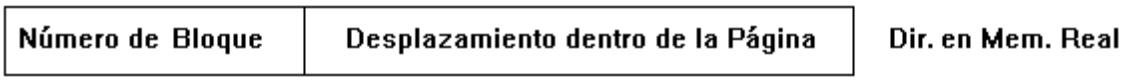


Fig. 13.12.

Por ejemplo :

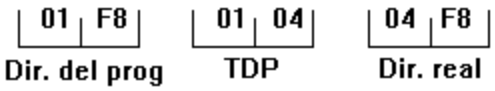


Fig. 13.13.

Las tablas están en memoria, luego para obtener una dirección efectiva a cualquier posición de un programa se torna necesario acceder dos veces a memoria : la primera para llegar a la TDP y obtener la dirección del bloque correspondiente y la segunda para llegar efectivamente a la dirección deseada.

Una de las soluciones posibles a este doble acceso es cargar la TDP en el procesador en registros (se cargarían juntamente con la PC y los registros en el momento de cambio de contexto), pero esto limita mucho la cantidad de páginas que puede tener un programa y es además muy costoso.

Una solución más económica es el uso en memoria CACHE que contiene todas o una porción de las entradas de la tabla. Por el mismo algoritmo de actualización de la cache nos aseguraríamos que con el transcurrir del tiempo las entradas que permanecerán en cache serán las más usadas.

Como mecanismos **hardware** nuevos se agregan el Traductor de Direcciones (DAT), los registros de páginas o la memoria Cache y como mecanismo de protección se suele utilizar un bit (o clave) de protección asociado a cada bloque.

La clave de protección será usado para todos los accesos a memoria que no se realicen por medio del DAT.

Como rutinas de **software** se agregan en esta administración las tablas de páginas por programas y las rutinas de manejo de dichas tablas.

13.9 - **MEMORIA VIRTUAL**

Definimos como **Espacio de Direccionamiento de un Sistema** (computadora) a la capacidad de direccionamiento de la misma, o sea la dirección a la que puede llegar con todos los bits de direccionamiento en ON, por ejemplo de 0 a mm.

Definimos como **Espacio de Direccionamiento de un Proceso** a la capacidad de direccionamiento del mismo, o sea que pueda generar o pedir direcciones de 0 a nn.

Obviamente para que este proceso ejecute en este sistema debe ser nn £ mm.

Por otra parte ya sabemos que no necesariamente la memoria disponible en una computadora es igual a su espacio de direcciones, sino que generalmente es más chica, e inclusive puede ser menor que el espacio de direccionamiento requerido por un proceso.

Muchas veces es necesario escribir programas cuyo tamaño supere el tamaño de la memoria real existente en la instalación.

A raíz de esta necesidad surge lo que denominaremos MEMORIA VIRTUAL que simula poseer una cantidad de memoria mayor que la existente, y a continuación se tratan los métodos de administración para este tipo de memorias.

Usualmente en Memoria Virtual la memoria aparece al programador tan grande como él la necesite, existiendo en este caso un mecanismo provisto por el método de administración que se encarga de poner a disposición del procesador los fragmentos (tanto páginas como segmentos) que sean requeridos en el momento de que sucede tal requerimiento.

Se debe tener en cuenta que si un proceso utiliza para si todo el espacio de direcciones de un sistema, mientras él esté ejecutando será el único proceso en ejecución (lo que comúnmente se llama Espacio único de direcciones).

Ahora bien, si se puede simular, en las técnicas que se verán más adelante, la posibilidad de poseer mayor memoria real que la que existe, porqué no es posible simular también que existen más espacios de direcciones ? (lo que comúnmente se llama Espacio múltiple de direcciones).

Esto se puede lograr independizando completamente las direcciones de los procesos de las direcciones de memoria real, permitiendo que cada proceso posea su propio juego de direcciones, o sea permitiendo que existan tantas direcciones 00 ó 100 como procesos existan. Esta es una conclusión casi trivial cuando se observa la construcción de las Tablas de Distribución de Páginas.

13.10 - **Administración PAGINADA POR DEMANDA**

Para ver el funcionamiento de este tipo de administración debemos contar primero como llega un programa al sistema.

Originalmente el programa se encuentra en una biblioteca (usualmente en algún periférico tipo disco magnético) desde la cual es invocado para su ejecución.

El programa se carga entonces en otro periférico (aquí también usualmente discos magnéticos) en donde es preparado para la ejecución (se arma su BCP, se completa su TDP, se le agrega información respecto a buffers de archivos, etc.).

Una vez que el programa está listo se carga su TDP en memoria real y se pasa al proceso a estado de Listo en espera del control de la CPU.

Una peculiaridad que existe en esta administración es que en la TDP se agrega un bit que indica si la página que se está referenciando existe en memoria real o no y además se agrega un campo que indica la dirección de tal pagina en el dispositivo de memoria virtual.

Los programas con el transcurrir del tiempo no se cargan en su totalidad en memoria real, sino que se cargan aquellas páginas que van siendo solicitadas para su ejecución.

Para su administración se necesitan un par de tablas cuyos contenidos iremos analizando.

Tabla de Distribución de Páginas (una para cada proceso)

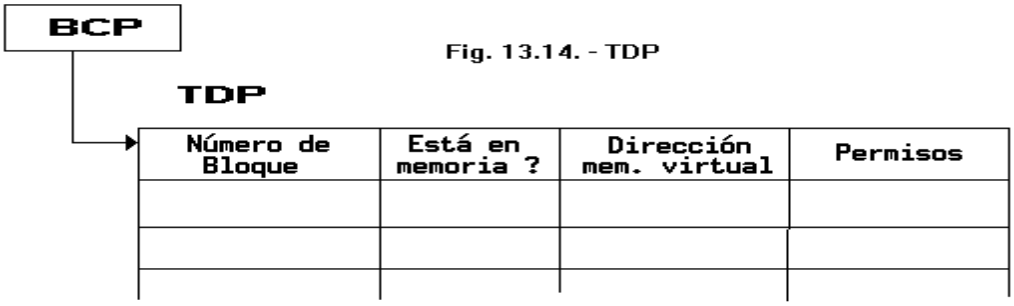


Fig. 13.14. - TDP

Tabla de Distribución de Bloques (una en todo el sistema)

Cuando se encuentra encendido el bit que indica que la página no se encuentra en memoria real se produce una interrupción por falta de página. La rutina que atiende esa interrupción necesita la dirección de memoria

TDB

Id. del programa	Página	Contador	Página cambió ?	Fijo por canal ?	Tránsito?

Fig. 13.15. - TDB

virtual en donde se encuentra almacenada dicha página para poder traerla a memoria real.

Supongamos que la dirección del lugar en donde se encuentra la página en memoria virtual se encuentra a nivel del BCP en lugar de figurar en la TDP para cada una de las páginas que componen ese programa, es decir que existe una única dirección que apunta a donde se encuentran almacenadas en memoria virtual las páginas de este programa.

La conclusión inmediata es que esta forma nos acarreará un problema de administración respecto de la contigüidad del programa en memoria virtual y además hará sumamente engorroso el ubicar una página determinada ya que llevaría a la lectura secuencial de todas sus páginas (comenzando desde la dirección apuntada desde el BCP).

La otra manera, o sea, cada entrada en la TDP tiene su propio apuntador a memoria virtual, permite mantener en forma discontigua a los programas en disco y acceder en forma directa a la página buscada.

El bit de cambio que se encuentra en la TDB es necesario para toda vez que sea imprescindible remover una página de memoria real con el objeto de traer otra, ya que si la página que se va a remover sufrió modificaciones debe ser regrabada en memoria virtual para preservar la integridad de la aplicación.

El bit de Fijo por canal se utiliza para fijar páginas en memoria real; esto es, cuando una página contiene información que está siendo dirigida o está recibiendo información de un canal de E/S, debido a que los canales deben utilizar direcciones absolutas se torna imprescindible fijar tales páginas en memoria real. Es decir no pueden ser removidas. Nótese que también se encuentran en esta situación aquellos bloques de memoria real sobre los cuales se está cargando una página o de los cuales se está descargando una página en memoria virtual. Normalmente a estas últimos bloques suele denominárselos "en tránsito".

13.10.1 - **Interrupción por PAGE FAULT.**

El mecanismo de direccionamiento opera igual que en el método de administración de memoria paginada simple.

Si el procesador intenta direccionar a una página que no se encuentra cargada en memoria real se produce lo que se denomina una **Interrupción por falla de página (Page-Fault)**.

Podemos graficar el mecanismo completo de direccionamiento incluyendo el procesamiento de la interrupción de la falla de página como se puede visualizar en la Fig. 13.16.

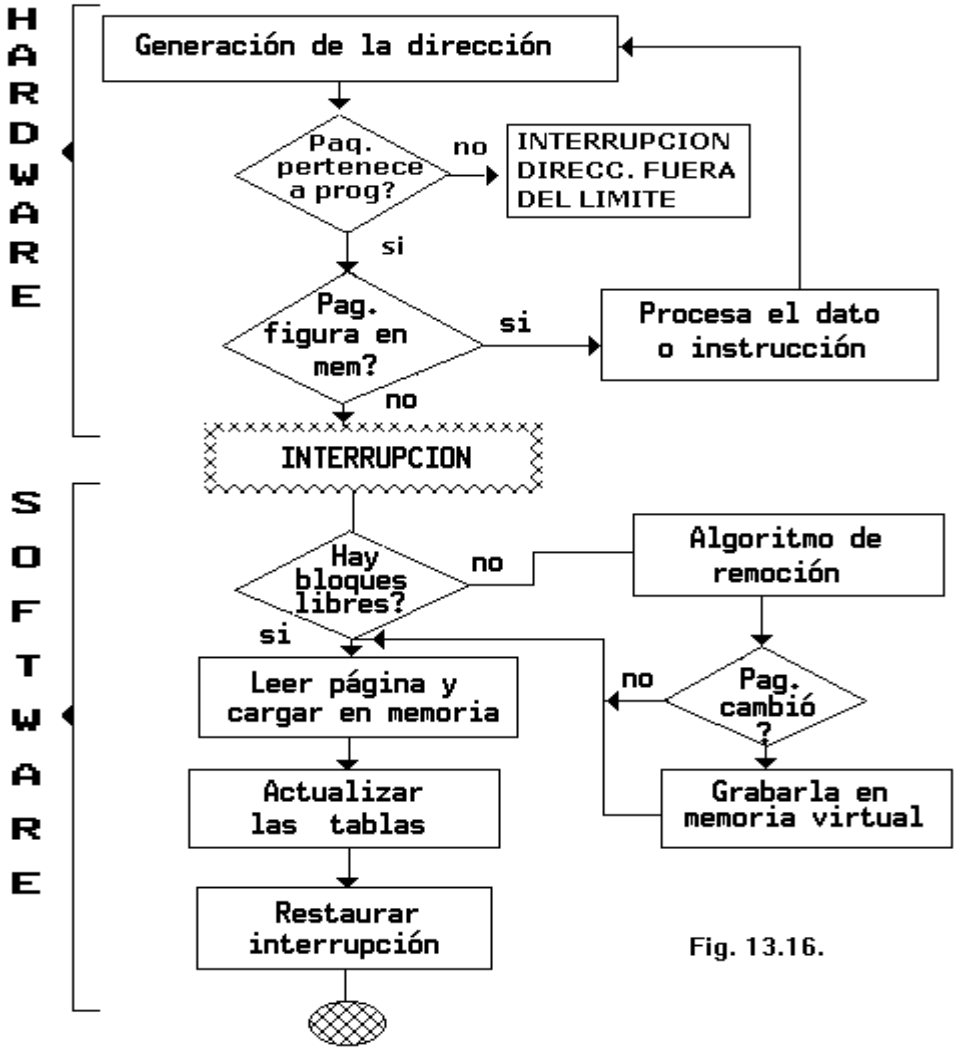


Fig. 13.16.

Desde la generación de la dirección hasta la interrupción inclusive, todos los pasos se realizan por **hardware**. En cambio desde el momento en que se comienza a ejecutar la rutina que atiende esa interrupción hasta la restauración final del estado del estado previo antes de la interrupción se realiza por **software**.

Si la interrupción se produce a causa de una dirección generada por el RPI no hay mayor problema, pues no se había comenzado la ejecución de la instrucción.

En cambio, si la interrupción se produce como consecuencia de una dirección generada por los operandos de una instrucción que está ejecutándose se nos plantea un problema ya que nos encontramos en medio de la ejecución.

Existen dos caminos posibles :

- Recomenzar (restart) : Al producirse la interrupción se restaura la instrucción al estado inicial (previo a su ejecución). Hay que volver también los registros y posiciones de memoria cambiados a su estado anterior, por lo tanto esto nos obliga a llevar una historia de la instrucción. Una implementación sencilla que nos ahorraría el mantener esta historia sería hacer antes que nada la verificación de las direcciones de los operandos (o sea antes de modificar nada) (Esta técnica acompaña muy frecuentemente a un Pipeline).
- Continuación : Se salva el estado de la instrucción y luego de satisfecha la interrupción se continúa la instrucción en el punto en donde se había abandonado. Esto implica salvar la información a nivel de micro-instrucción (salvar registros temporarios, estado de la instrucción, etc.). No todo tipo de instrucción puede soportar esto (pensemos por ejemplo en la instrucción Test & Set), ya que se ven obligadas a que en caso de ser interrumpidas a Recomenzar, por lo tanto se hace necesario el aplicar distintos mecanismos según el tipo de instrucción.

13.10.2 - **Traza**

Definiremos como **traza** de un programa a la enumeración de las páginas que ese programa referencia a medida que se ejecuta.

Definiremos también la siguiente relación :

f = número de páginas traídas / número de referencias de la traza

siendo f el **Indice de Fracasos** y si calculamos

s = 1 - f

llamamos a s el **Indice de Hallazgos**.

Lógicamente estos valores sufren una sustancial modificación dependiendo de los bloques libres que existen en memoria real sobre los cuales se pueden cargar páginas desde memoria virtual.

Es por tanto necesario antes de calcular "s" o "f" conocer la cantidad de bloques libres en memoria real.

La carga de la primer página de la traza implica también un fracaso ya que tanto si es invocada por el programa al ejecutar su primera instrucción o por que el Planificador de Trabajos le carga la primer página al proceso a ejecutar ambas situaciones implican una falla de página.

13.10.3 - **Algoritmos de Remoción**.

Llegado el momento que la tabla de distribución de bloques indica que no existe ningún bloque libre para asignar a una página que se está solicitando se hace necesario por tanto eliminar una página de memoria real. Denominamos a esto **Remoción**, y existen diversas políticas de selección de la víctima.

El mecanismo **óptimo** sería seleccionar aquella página que se tardará más en volver a usar, pero lamentablemente esto es sumamente difícil de predecir.

Es entonces a partir de aquí que se comenzará a utilizar el campo Contador de la TDB.

Otro mecanismo es el **FIFO** (first-in first-out), en donde la página que se selecciona para remoción es aquella más antigua en el sistema.

El método FIFO produce en determinados casos lo que se denomina "Anomalía de Belady". Supongamos un trazado del siguiente tipo :

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

si se realiza el cálculo de los indices de hallazgos y de fracasos para un tamaño de memoria de

M = 3 bloques y

M = 4 bloques

se obtiene que en el caso en que se tienen 4 bloques libres (es decir que se ha incrementado la cantidad de memoria libre que se puede asignar) se obtiene un índice de fracasos mayor que en el caso de M = 3.

Como conclusión se puede decir que a partir de un determinado punto el incrementar la memoria disponible no mejora el índice de fracasos e inclusive en determinados casos hasta lo empeora.

Una mejora que se puede introducir al algoritmo FIFO es el de otorgar una **segunda chance**, mecanismo mediante el cual si la página es referenciada una segunda vez se pone a cero un bit que la acompaña. Por tanto esa página tiene otra oportunidad para permanecer en memoria y no ser seleccionada para remoción.

Otro método de remoción es el **LRU (Least Recently Used)**. En este método aquella página que ha sido menos recientemente utilizada es la que se selecciona para remoción.

Lógicamente para lograr este fin el sistema debe llevar la información necesaria para poder determinar quién es la víctima. Esto se logra mediante bits que se colocan a cero o a uno dependiendo de cuanto tiempo hace que la página fue referenciada.

Otro algoritmo es el **LFU (Least Frequently Used)**, en este caso la página menos usada es la elegida para remoción. Para poder lograr esto el sistema lleva una cuenta sobre cuantas veces la página fue referenciada.

Otro método consiste en no solo llevar información respecto a las referencias realizadas a la página, sino también utilizar la información del bit de cambio, ya que en aquellos casos en que la página que se selecciona para remoción ha sido alterada es absolutamente imprescindible su regrabación en memoria virtual.

13.10.4 - **Cuestiones de implementación**.

En general el tamaño de la página coincide exactamente con el tamaño del bloque, aunque existen implementaciones en las que el tamaño de la página es un múltiplo del tamaño del bloque.

A efectos de simplificar nuestro trabajo asumiremos en nuestra materia que el tamaño de la página coincide siempre con el tamaño del bloque.

En este sistema de administración de memoria existe fragmentación, ya que generalmente la última página del programa no ocupa exactamente un bloque. Sin embargo la fragmentación solo llega a lo sumo al tamaño de una página por cada programa dentro del sistema.

Un problema bastante frecuente en esta implementación de administración de memoria es el buffer que cruza el límite de una página a otra, usualmente denominado "buffer a caballo".

Afortunadamente los canales pueden manejar esta eventualidad utilizando para ello programas de canal concatenados que permiten transferir parte de la información a un sector de memoria real y continuar la transferencia sobre otras direcciones de memoria. Recuérdese que en estos casos ambas páginas deben estar fijas en memoria.

Cuando el algoritmo de remoción es muy malo o en aquellos casos en que la memoria real es drásticamente más pequeña que la memoria virtual se puede producir un fenómeno denominado **thrashing**.

Thrashing es el estado en que se encuentra un sistema cuando esta realizando un excesivo intercambio de páginas, en otras palabras, el sistema pasa más tiempo ejecutando las rutinas de remoción y carga de páginas que en ejecutar las instrucciones de los procesos propiamente dichos.

Como mecanismos nuevos de **hardware** se agregan en este tipo de administración : los mecanismos de protección por bloques (que no son utilizados por el DAT), la interrupción por falla de página, los contadores de uso de las páginas, los bits de páginas cambiadas y la fijación de páginas por el canal de E/S.

Como elementos nuevos de **software** tenemos aquí la Tabla de Distribución de Páginas, las rutinas de atención de interrupciones por falla de página, los algoritmos de remoción, la rutina de búsqueda de páginas, la de grabación de páginas, las rutinas de actualización de las tablas en memoria y las rutinas de administración de la memoria virtual.

13.11 - **Administración de Memoria con SEGMENTACION**.

Número de segmento	Desplazamiento en el segmento

Fig. 13.17.

La administración de memoria segmentada puede darse con o sin memoria virtual, aquí la veremos con memoria virtual.

En este esquema la división de la memoria se produce según el tamaño de los segmentos.

Un **segmento** es una unidad lógica del programa; por ejemplo : una división lógica, un área Common (en Fortran), un vector, una Section del Cobol, una subrutina, etc.

Necesitamos también tablas de administración de segmentos que son mayores que las tablas de páginas ya que aquí los segmentos difieren en longitud, y por tanto debemos guardar para cada segmento no solo su ubicación en memoria sino su tamaño. (Implementado en equipos tales como el Burroughs B5500 y el Honeywell

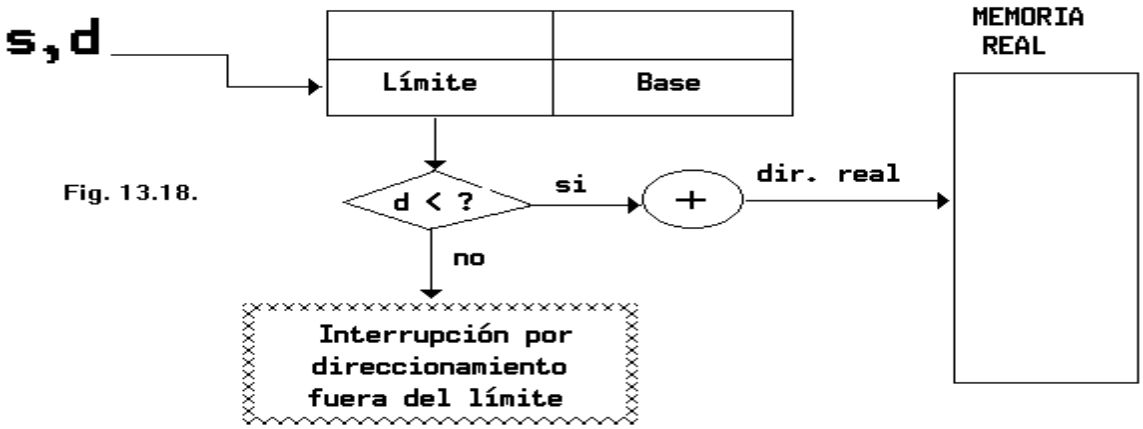


Fig. 13.18.

6180 bajo el sistema operativo MULTICS).

Cada dirección se encuentra desdoblada en número de segmento y desplazamiento (offset) dentro del segmento (S,d) :

Ya que para cada segmento se tiene su dirección de origen (base) y su longitud (límite), el mecanismo de direccionamiento opera de la forma que puede visualizarse en la Fig. 13.18.

Los segmentos también pueden ser compartidos, para lo cual deben ser reentrantes y existen además permisos de acceso asociados a ellos.

Tales segmentos se vinculan solamente en el momento de ser necesarios y tal vinculación se realiza en memoria virtual. Esto nos lleva a una característica de este tipo de administración de memoria que se denomina "**vinculación dinámica".**

Cuando un programa se compila en este tipo de sistemas de administración no se resuelven todas sus direcciones en tiempo de vinculación, ya que de existir referencias externas estas se resuelven solamente en tiempo de ejecución. Esto permite justamente el poder referenciar a segmentos compartidos por varios usuarios y no duplicar innecesariamente copias del mismo en memoria real.

Existe fragmentación también en este tipo de administración debido a las áreas libres que quedan en memoria real y que no pueden ser asignadas a ningún segmento, un poco como la situación planteada en la administración Particionada Variable. Y aquí también puede existir algún mecanismo de Compactación.

13.11.1 - **Tablas necesarias**.

Las tablas que se requieren en este tipo de administración son cuatro, a saber :

- Tabla de Mapa de Segmentos (existe una por cada espacio de dirección).
- Tabla de Areas no Asignadas (existe una sola en el sistema).
- Tabla de Nombres de Segmentos (existe una por cada espacio de dirección).
- Tabla de Estado de Segmentos Activos (una en todo el sistema).

Una implementación posible de estas tablas puede ser visualizada como se ve en la Fig. 13.19, por razones de simplicidad pondremos a las tablas de Mapas de Segmentos y Nombres en una sola.

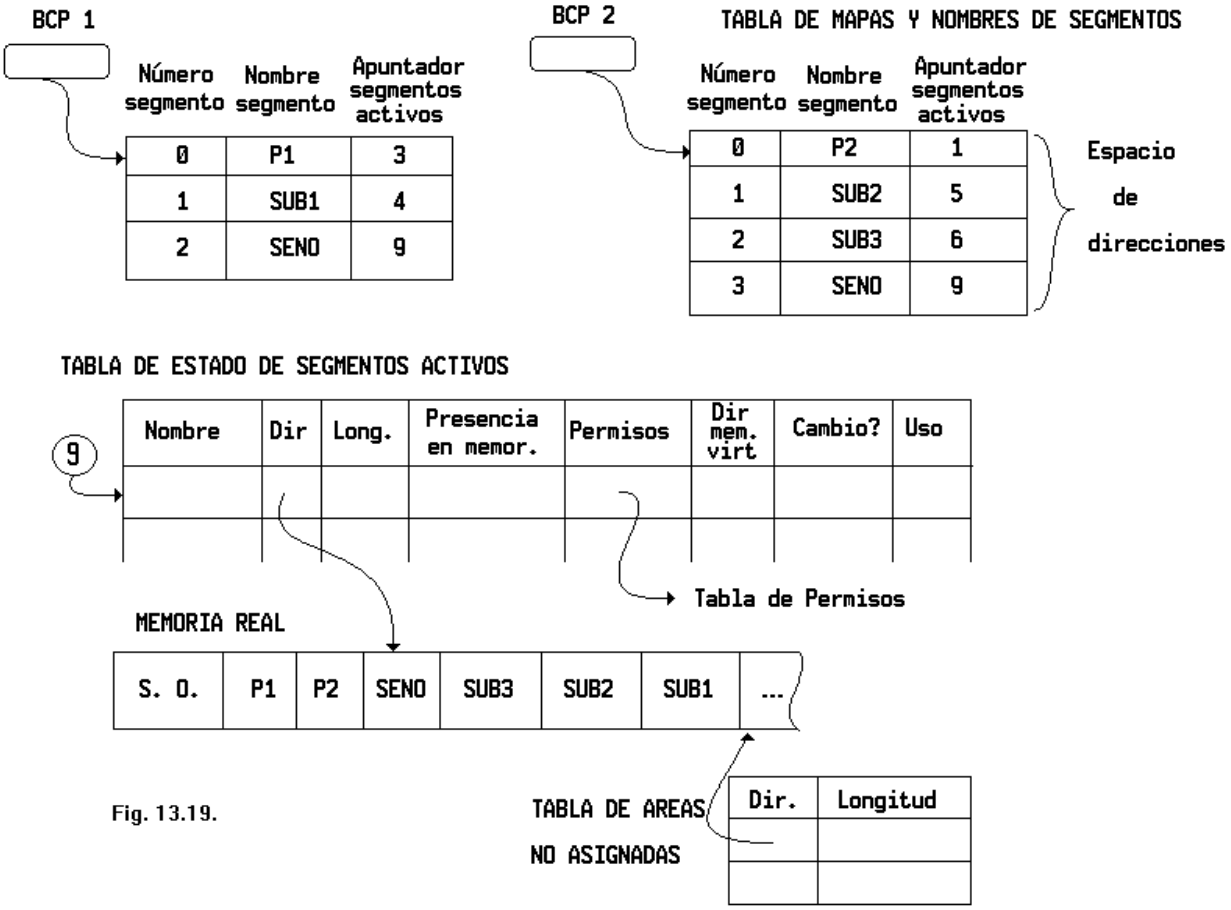


Fig. 13.19.

13.11.1.1 - **Mecanismo de trabajo de las tablas**

El mecanismo de trabajo de las tablas sería, por ejemplo, que si desde el proceso P2 se solicita la utilización de la subrutina Sub2, (o sea el segmento1), se busca dentro de la tabla de Mapa y Nombres de Segmentos lo solicitado, una vez hallado el dato se toma como información el apuntador a la Tabla de Estado de Segmentos

Activos, de la cual se obtiene la dirección del segmento buscado y sumándole el desplazamiento al que se desea acceder se obtiene la dirección real en la cual se encuentra el dato o instrucción deseados.

Podemos asimilar, en principio, este tipo de manejo, al de paginación, con la fuerte observación de que los segmentos son todos de longitud distinta, o sea tendrán un tamaño de acuerdo a una razón lógica, luego es muy importante conocer su dirección de comienzo y su longitud, elemento que se deberá utilizar para no permitir el acceso a direcciones no propias.

El campo Permiso, que generalmente será un apuntador a una tabla asociada, indicará las formas de acceso a ese segmento desde los distintos espacios de direcciones, identificados por sus BCP. Este campo está dado principalmente pues en este sistema se realiza con frecuencia la compartición de segmentos, para evitar la proliferación de un mismo código en función de las veces que es invocado. Obviamente estos códigos deberán ser reentrantes cuando sean compartidos.

Veamos cuál es el manejo de las tablas cuando se da la situación de que se invoca un segmentos que nunca antes habían sido invocado.

Supóngase que Sub2, invoca, por primera vez (para él), la subrutina SENO.

Si es la primera vez, significa que aún no tiene resuelta sus direcciones para ese segmento (Ver más adelante resolución de direcciones). Como aún no está resuelta, significa que se generará una interrupción para ese efecto.

La rutina que recibe esta solicitud, revisará la propia Tabla de Mapas y Nombres de Segmentos, con el parámetro SENO, al encontrarla, identificará ese segmento como el número 3 y devolverá esa información al segmento Sub2, con lo cual se realizará la vinculación para esta sesión.

En caso de haber solicitado el acceso a un segmento que no se encontrara en su propia Tabla de Mapas y Nombres de Segmentos, se debería buscar en la Tabla de Segmentos Activos, si allí se hubiese hallado, se consultaría el Permiso correspondiente, que en caso de ser válido provocaría que en la Tabla de Mapas y Nombres de Segmentos se generara una entrada para este nuevo segmento con el número correspondiente para este espacio de direcciones y se procedería a su posterior vinculación. Si el permiso hubiese sido no válido se adoptaría algún mecanismo de espera o cancelación.

Si el segmento tampoco hubiese sido hallado en la Tabla de Segmentos Activos, eso significa, que ninguno de los procesos actualmente en ejecución lo invocó, en consecuencia, es necesario generar una entrada en la

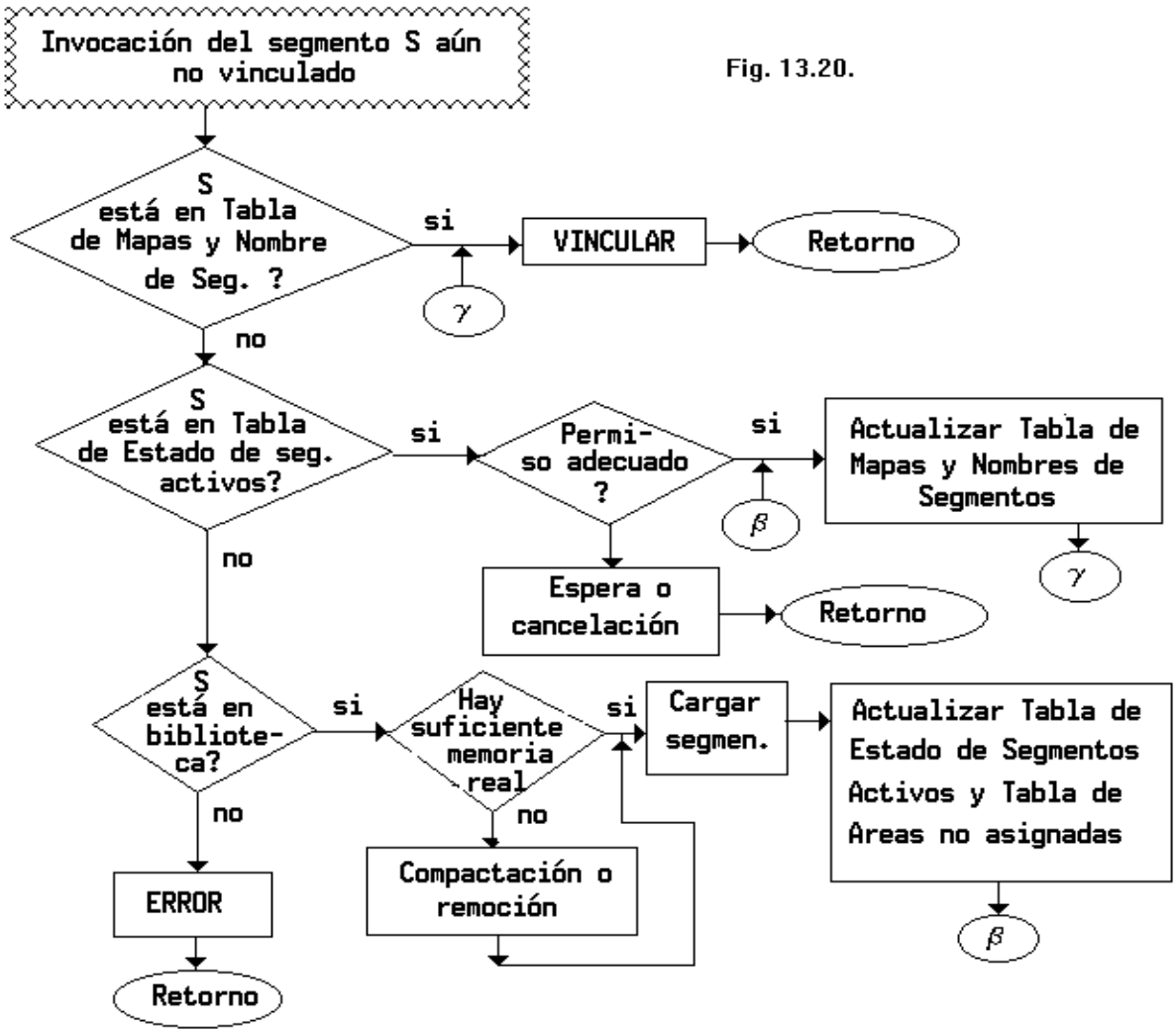


Fig. 13.20.

Tabla de Segmentos Activos, buscar el segmento en biblioteca, verificar la posibilidad de cargar el segmento en memoria consultando la Tabla de Areas no Asignadas, y si es posible cargarlo, completar los datos de la Tabla de Segmentos Activos, actualizar la Tabla de Mapas y Nombres de Segmentos del solicitante y vincular.

En caso de que no exista espacio suficiente en memoria real será necesario aplicar compactación o remoción de otros segmentos.

Luego, la actividad de búsqueda de segmentos aún no vinculados se puede diagramar como se ve en la Fig. 13.20.

13.11.2 - **Encadenamiento de Segmentos o Vinculación Dinámica**

Ya se mencionó que la vinculación se realiza durante la ejecución. Para poder cumplir esto se dice que el direccionamiento a otros segmentos se realizará de forma indirecta.

O sea que un llamado del tipo CALL SUB1 se transformará, por intervención del compilador en la siguiente pieza de código:

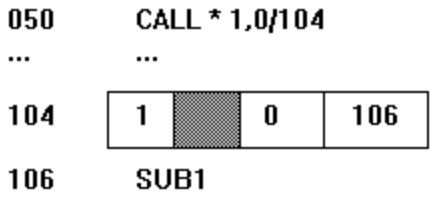


Fig. 13.21.

Donde:

*	indica direccionamiento indirecto.
1	registro con dirección de retorno
0/104	dirección de dirección

El bit en 1 en 104 indica dirección no resuelta

0/108	dirección del nombre del segmento
SUB1	nombre del segmento

Si durante toda la ejecución de este segmento no se ejecuta la instrucción que se encuentra en la dirección 050, SUB1 nunca será invocada, ni llamada, ni vinculada.

Si por el contrario se ejecuta el CALL, se irá a la dirección 104, donde el bit en 1 indicará dirección no resuelta y esto provocará la interrupción, Una vez resuelta ésta en la dirección 104 quedará:

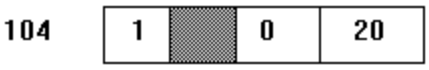


Fig. 13.22.

donde:

0	indica dirección resuelta
1	indica segmento 1
20	indica instrucción ejecutable (Entry Point)

Nota: el pasaje de parámetros es una convención, la cual puede ser Stack, registros, etc.

Hasta ahora hemos supuesto que las direcciones de los datos e instrucciones eran resueltos tomando la dirección del segmento y sumándole el desplazamiento, o sea que frente a una dirección S,d , se busca dentro de la Tabla de Segmentos el valor S, se toma su dirección y se le suma d, pero, qué sucede cuando un segmento se referencia a sí mismo, o cuando es compartido, siendo referenciado desde distintos espacios de direcciones con número de segmentos distintos ?

Obviamente no es posible cambiar en todas sus direcciones los números de segmento, y si además es compartido cada espacio de dirección lo ve como un número de segmento distinto, lo que obligaría a que dependiendo del llamador se cambiase el número de segmento.

Existen algunas propuestas, como por ejemplo, que todos los segmentos compartidos deberían tener un número de segmento tan alto, que ningún espacio de dirección pudiese llegar a ese número (Peterson 1985).

Esto tiene algunas desventajas, como ser que habría que conocer de antemano todos los números de segmentos, y si las entradas de las Tablas de Nombres y Mapas de Segmentos son correlativas, habría que guardar muchas entradas no utilizadas para llegar a esos números, con la consiguiente pérdida de espacio en memoria.

Otra propuesta es tener en claro cuando se direcciona dentro del segmento y cuando se direcciona a otro ó sea fuera del segmento. El Multics GE-645 implementó un indicador de dos estados (ON- OFF), estableciendo que si el indicador está en OFF se está direccionando dentro del segmento y la dirección base del segmento se encuentra en el Registro Base (Procesador) previamente cargada. Si está en ON, indica que la dirección se obtiene como S + d, si es búsqueda de dato, si se ejecutan instrucciones será necesario un cambio de Registro Base

(por ejemplo BIF 3/120 implica cambio de Registro Base). Nota: es mandatorio que los retornos siempre se realicen como retornos de subrutinas según la convención que se establezca y que esto implique la carga del Registro Base del Segmento llamador.

Una implementación sencilla es que cada segmento se ve a sí mismo como el segmento 0 (cero) (OFF). Nota: debe existir un registro de los posibles Registros Bases, que puede ser adicionado al BCP, o Tabla de Mapas y Nombres de Segmentos.

13.12 - **Administración de SEGMENTACION PAGINADA**

Esta administración junta las ventajas y desventajas de ambas administraciones. Comparte segmentos, aprovecha el uso de la paginación por demanda, con lo cual existen 2 tipos de interrupciones, falta de segmento y falta de página.

Debido al tamaño de las tablas es posible necesitar 3 accesos a memoria por cada acceso que se desee realizar, ya que es difícil que las tablas puedan ser contenidas en almacenamientos del procesador.

Su esquema puede verse en la Fig. 13.23.

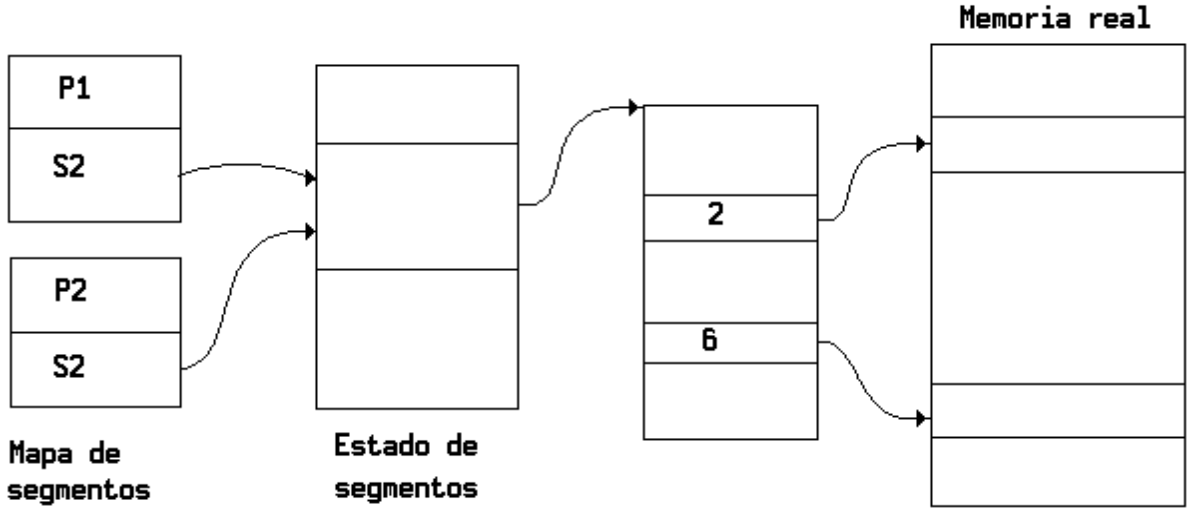


Fig. 13.23. - Administración de Segmentación Paginada.

Las necesidades de Hardware son:

- Protección
- Dirección Base
- Traducción de Direcciones
- Registro de Reubicación.

Las necesidades Software son :

- Tablas de Segmentos
- Dirección Base
- Longitud
- Presencia
- Cambio
- Uso
- Permisos
- Dirección Memoria Virtual
- Rutinas de atención por falta de Páginas
- Algoritmos de remoción
- Rutinas de Búsqueda de Páginas y Segmentos
- Rutinas de Vinculación en ejecución.