Tema 7: Memoria Virtual.

Objetivos:

- Analizar la necesidad de introducir el mecanismo de memoria virtual en un computador.
- Estudiar el funcionamiento de la memoria virtual paginada y las alternativas de diseño para la tabla de páginas, políticas de búsqueda y políticas de sustitución.
- Estudiar el funcionamiento de la memoria virtual segmentada y sus políticas de sustitución de segmentos.
- Introducir la memoria virtual con segmentos paginados como una alternativa de síntesis de las dos anteriores
- Analizar el sistema de memoria virtual de algunos procesadores.

Contenido:

1. Gestión de memoria

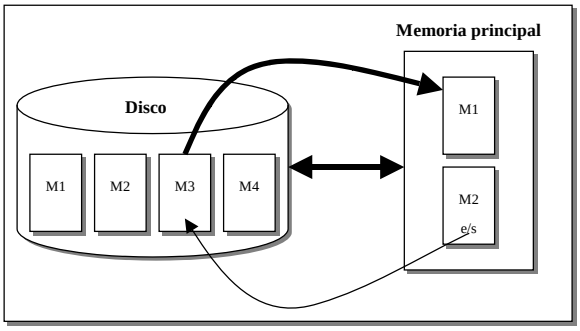
El sistema de memoria virtual de los actuales computadores surgió para liberar al programador de una serie de tareas relacionadas con el uso que los programas debían realizar con la memoria. La memoria virtual automatiza la gestión entre los dos niveles principales de la jerarquía de memoria: memoria principal y disco. Antes de entrar en los mecanismos específicos de la memoria virtual revisaremos una serie de funciones que deben incorporarse en la gestión de memoria.

1.1. Solapamiento (*overlay*)

El tamaño de la memoria principal disponible en los computadores ha aumentado de forma sostenida desde sus orígenes. Sin embargo, el tamaño de los programas ha crecido más rápidamente, por lo que la necesidad de ejecutar programas que no cabían en la memoria principal ha sido una constante en la historia de los computadores. Una forma de superar esta limitación es el uso de la técnica de solapamiento (*overlay*).

Esta técnica divide en módulos el programa cuyo tamaño sobrepasa la capacidad de la memoria principal, y que reside por tanto en memoria secundaria (disco). Después se introducen en los lugares adecuados de cada módulo, y al margen de la lógica propia del programa, las instrucciones de E/S necesarias para cargar en memoria principal aquellos módulos cuyas instrucciones deban ejecutarse o cuyos datos vayan a ser referenciados en el inmediato futuro. Es decir, el propio programa se ocupa de cargar por anticipado los módulos que van a ser referenciados.

Con este mecanismo se puede superar la limitación del tamaño de la memoria principal, pero tiene el inconveniente de hacer depender el programa de las dimensiones concretas de la memoria del computador para el que se codifica, obligando a revisar la división modular del programa cuando cambie la configuración de la máquina.



1.2. Reubicación

En sistemas con multiprogramación se necesita que varios programas residan simultáneamente en memoria. El tiempo de CPU se va distribuyendo entre ellos de acuerdo a una política de prioridades determinada. La ubicación en memoria de los programas no se conoce en tiempo de compilación, por lo que no se pueden generar direcciones absolutas. Para conseguir una asignación dinámica de memoria en tiempo de ejecución se utilizan *registros de reubicación*. La dirección efectiva se obtiene sumando a la dirección generada por el compilador el contenido del registro de reubicación asignado al programa.

1.3. Paginación

La paginación también surgió de la necesidad de mantener más de un programa residente en memoria cuando la capacidad de ésta es inferior a la suma de los tamaños de los programas. Se trata de un mecanismo automático de solapamiento múltiple que practica el Sistema Operativo para hacer posible la multiprogramación. El espacio de memoria principal se divide en bloques de tamaño fijo denominados páginas. Los programas se dividen también en páginas y residen en el disco. El Sistema Operativo se encarga de asignar páginas físicas a los programas en ejecución (multiprogramación). De esta forma el tiempo de CPU puede distribuirse entre los programas residentes.

1.4. Protección

Un papel importante de la gestión de memoria es la protección. Si varios programas comparten la memoria principal debe asegurarse que ninguno de ellos pueda modificar el espacio de memoria de los demás. Como casi todos los lenguajes permiten el uso de punteros dinámicos, los test en tiempo de compilación no son suficientes para garantizar la protección. Esta debe mantenerla en tiempo de ejecución el sistema de gestión de memoria (MMU).

1.5. Compartición

Esta función parece estar en contradicción con la anterior. Sin embargo, con frecuencia los programas de un sistema multiprogramado deben poder compartir y actualizar información, por ejemplo, un sistema de bases de datos. Además, no es necesario tener varias copias de una rutina si se permite que todos los programas accedan a una misma copia.

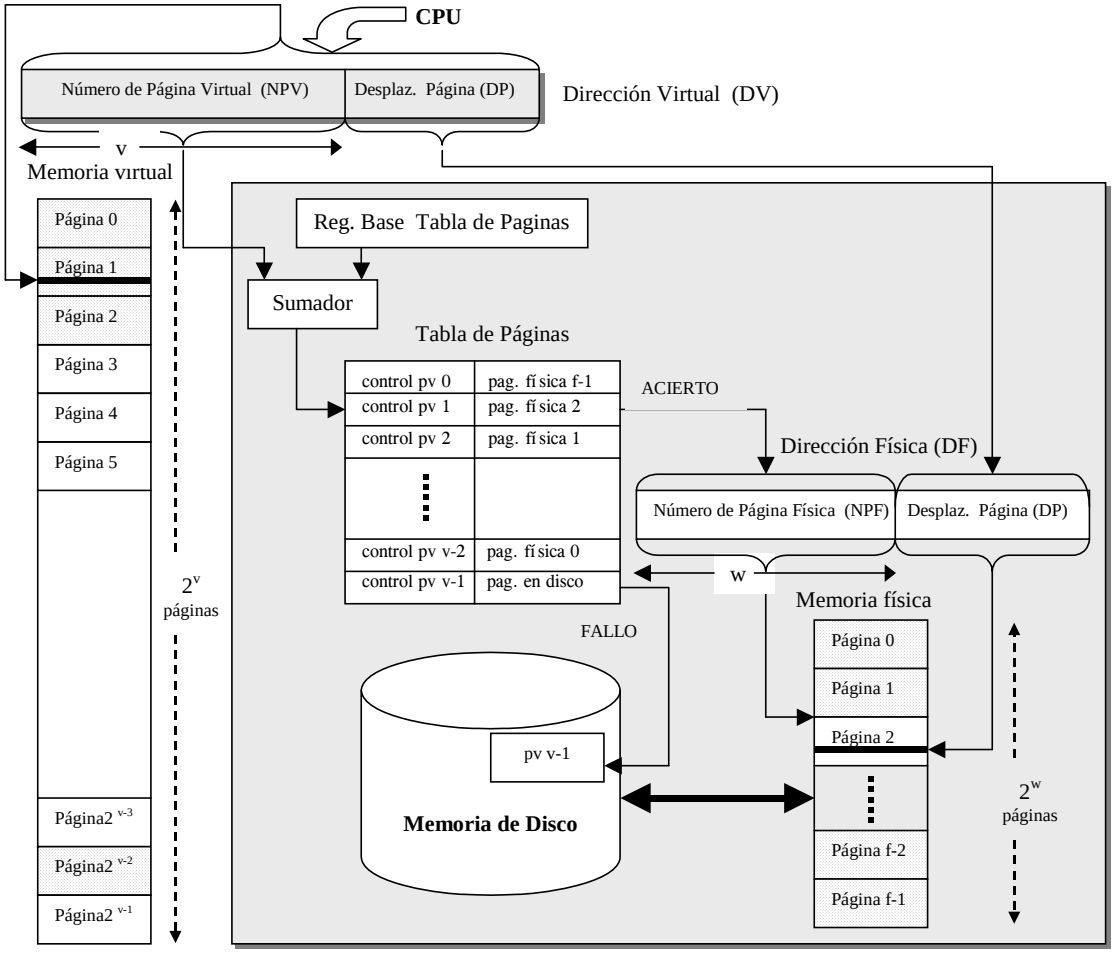
2. Memoria virtual

El sistema de memoria virtual implementa todas las funciones anteriores de forma integrada.

En un computador con memoria virtual (MV) las direcciones de los programas (generadas por la CPU) hacen referencia a un espacio mayor que el espacio físico realmente disponible en la memoria principal o memoria física (MF).

Los programas operan *virtualmente* con un tamaño físico de memoria principal mucho mayor que el realmente disponible. En estas máquinas hay que diferenciar, pues, entre el espacio de direcciones virtuales generado por la CPU y el espacio de direcciones físicas o reales existentes en memoria principal y determinado por el número de líneas del bus de direcciones.

El espacio virtual se soporta sobre un disco con la ayuda de un mecanismo de traducción que genera la dirección física de memoria principal a partir de la virtual. En la siguiente figura hemos representado el mecanismo de traducción de direcciones virtuales (DV) a direcciones físicas (DF).



Tanto la memoria principal como la memoria del disco se dividen en páginas de igual tamaño. El número de páginas de la memoria virtual en general es mayor que el número de páginas disponibles de la memoria física. Por este motivo, en cada momento sólo las copias de un conjunto de páginas virtuales del programa residen en la memoria física. Este conjunto recibe el nombre de *conjunto de trabajo* o *conjunto activo*, y resulta relativamente estable a lo largo del tiempo, debido a la localidad referencial que manifiestan los programas. Esta es la clave del buen funcionamiento de la memoria virtual, al igual que ocurría con la memoria cache.

Los bits de una DV se consideran divididos en dos campos, el *número de página virtual* (NPV) los más significativos, y el *desplazamiento dentro de la página* (DP), los menos significativos. El número de bits del campo DP lo determina el tamaño de página (*nº de bits de DP = log_2 tamaño*

de página). El número de bits del campo NPV lo determina el número de páginas virtuales (*nº de bits de NPV = log₂ nº de páginas virtuales*). Los bits de una DF se consideran divididos también en dos campos, el *número de página física* (NPF) los más significativos, y el *desplazamiento dentro de la página* (DP), los menos significativos. El número de bits del campo DP de una DF es el mismo que el de una DV, puesto que las páginas tienen igual tamaño en MV y MF. El número de bits del campo NPF lo determina el número de páginas físicas de MF (*nº de bits de NPF = log₂ nº de páginas físicas*).

Las DVs generadas por la CPU se traducen a DFs con la ayuda de una Tabla de Páginas (TP). Esta tabla contiene en principio tantas entradas como páginas existen en la MV, y la posición en la tabla de una entrada correspondiente a una página virtual concreta coincide con su NPV. Cada entrada contiene un primer campo de bits de control, de los que por el momento haremos referencia tan sólo a uno, el bit de presencia (P). Si este bit está activo (por ejemplo a 1) significa que la página virtual correspondiente a esa entrada está presente en la MF, y en este caso el segundo campo de la entrada contiene el correspondiente NPF. Si el bit P está inactivo (por ejemplo a 0) significa que la correspondiente página virtual no está en la MF, sino en el disco. En este caso el segundo campo de la entrada apunta a la dirección del disco donde se encuentra la página virtual. Como en un momento se pueden estar ejecutando más de un programa (multiprogramación o multiusuario) existirán más de una TP, una para cada programa (proceso) activo. Por ello, el acceso a la TP se realiza con la ayuda de un registro base de la tabla de páginas (RBTP) asociado a cada programa.

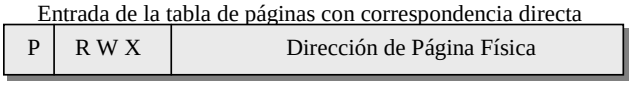
Para traducir una DV en DF se busca en la correspondiente entrada de la TP. Si el bit P de esta entrada vale 1, se dice que ha ocurrido un acierto de página, y se lee el contenido del segundo campo que en los aciertos constituye el NPF en la memoria principal. La DF completa se obtiene concatenando los bits de NPF con los de DP de la DV. Si el bit P de la entrada de la TP vale 0, se dice que ha ocurrido un fallo de página, lo que significa que la página virtual donde se ubica la DV que se está traduciendo, no se encuentra en MF. En este caso el fallo de página se sirve buscado la página en el disco, ubicándola en MF y actualizando la correspondiente entrada de la TP.

2.1. Memoria virtual paginada

El mecanismo de traducción de DV a DF que acabamos de describir corresponde a un sistema de memoria virtual paginada. En él el espacio virtual (y físico) se divide en páginas de igual tamaño. Veremos en el apartado siguiente otra alternativa en la que la MV se divide en segmentos de longitud variable, dando lugar a la memoria virtual segmentada. Las entradas de la TP de una MV paginada, además del NPF contiene unos bits de control, de los que hemos mencionado el bit de presencia P.

Como se muestra en la siguiente figura, existen otros bits que controlan los derechos de acceso a la página: lectura (R), escritura (W) y ejecución (X), este último sólo para páginas de código. También suele existir un bit que indica si la página ha sido modificada (datos) y necesita escribirse en disco cuando sea sustituida. Los demás bits de control dependen de cada procesador.

Entrada de la tabla de páginas con correspondencia directa



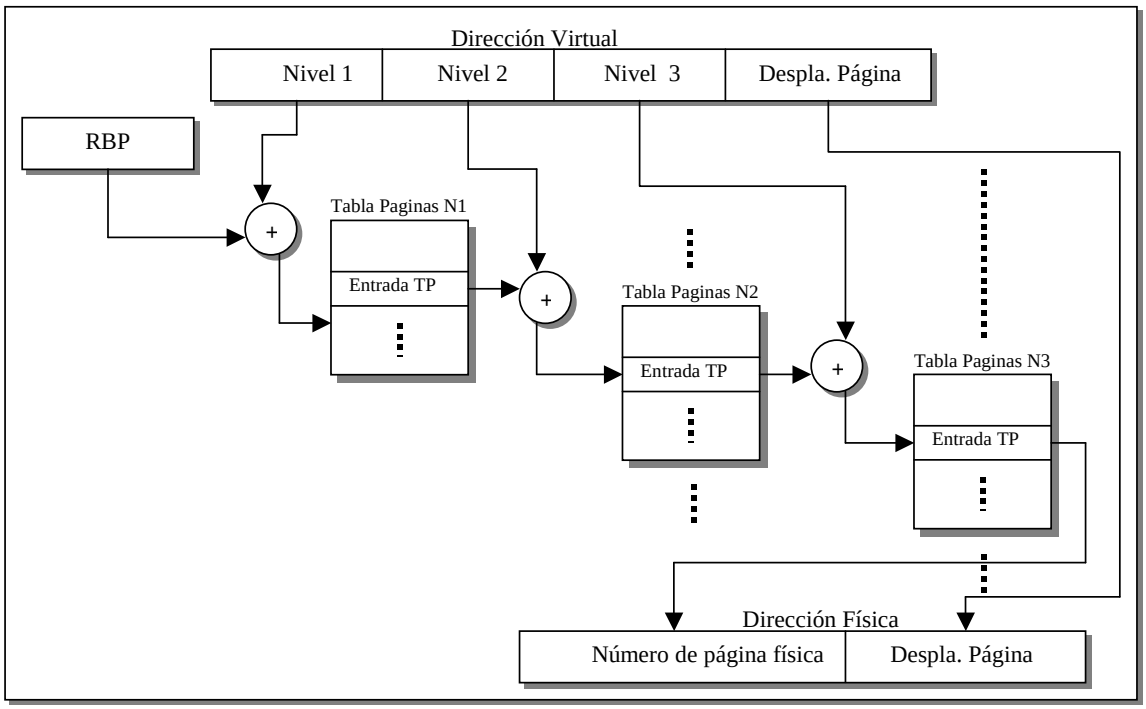
P = bit de presencia (de la página en memoria física)
RWX = derechos de acceso de lectura, escritura y ejecución

Cada programa (proceso) puede ocupar una gran cantidad de memoria virtual. Por ejemplo, en la arquitectura VAX, cada proceso puede tener hasta 2^{31} = 2 GBytes de memoria virtual. Utilizando páginas de 2^9 = 512 bytes, eso significa que se necesitan tablas de páginas de 2^{22} entradas por proceso. La cantidad de memoria dedicada sólo a tablas de páginas podría ser inaceptablemente alta.

Para solucionar este problema, la mayoría de los esquemas de memoria virtual almacenan las tablas de páginas en la propia memoria virtual, en lugar de utilizar la memoria física. Esto significa que la tabla de páginas también está sujeta a paginación, igual que el resto de los programas y datos. Cuando un programa se está ejecutando, al menos una parte de su tabla de páginas, incluyendo el elemento correspondiente a la página actualmente en ejecución, debe estar en la memoria principal.

2.1.1. Tablas de páginas de varios niveles

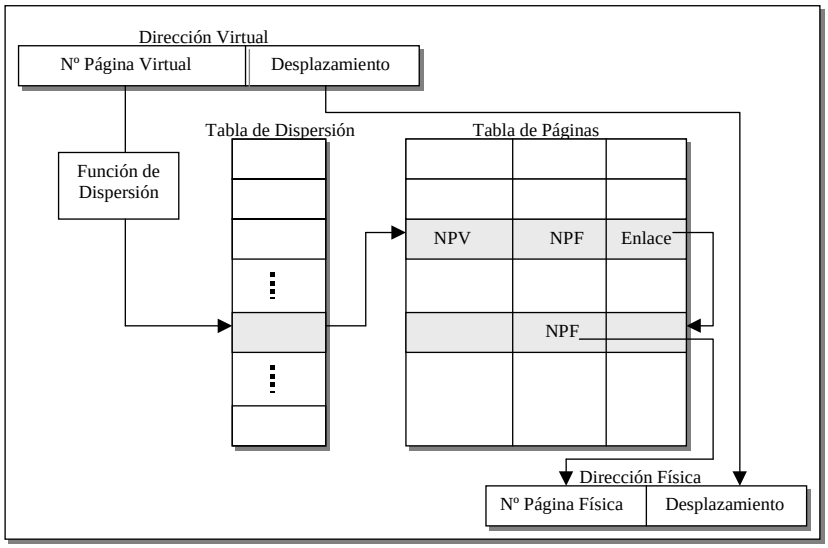
Algunos procesadores hacen uso de un esquema de dos o más niveles para organizar las tablas de páginas. En este esquema, hay una página de directorio de nivel 1 en la que cada elemento apunta a una tabla de páginas de nivel 2, y así sucesivamente. En la siguiente figura se muestra una TP organizada en tres niveles: N1, N2 y N3. Típicamente, la longitud máxima de una tabla de páginas se restringe al tamaño de una página.



2.1.2. Tabla de páginas invertida HASH

Esta alternativa elimina de la TP las entradas que no apuntan a una página de la memoria física, reduciendo las entradas a número igual al de páginas de la memoria física. El campo NPV de la DV se hace corresponder sobre una tabla de dispersión (*tabla hash*) mediante una función de dispersión sencilla. La tabla de dispersión incluye un puntero a una TP invertida, que contiene los elementos de la TP. Existe un elemento en la tabla de dispersión y en la tabla de páginas invertida para cada página de memoria física, en vez de para cada página de memoria virtual.

Se necesita, pues, una zona fija de la memoria física para las tablas, independientemente del número de programas o páginas virtuales que se admitan. Puesto que más de una DV puede apuntar al mismo elemento de la página de dispersión, se utiliza una técnica de encadenamiento para solucionar este problema. La técnica de dispersión da lugar a cadenas usualmente cortas, con uno o dos elementos. Esta alternativa de TP se utiliza en el PowerPC.

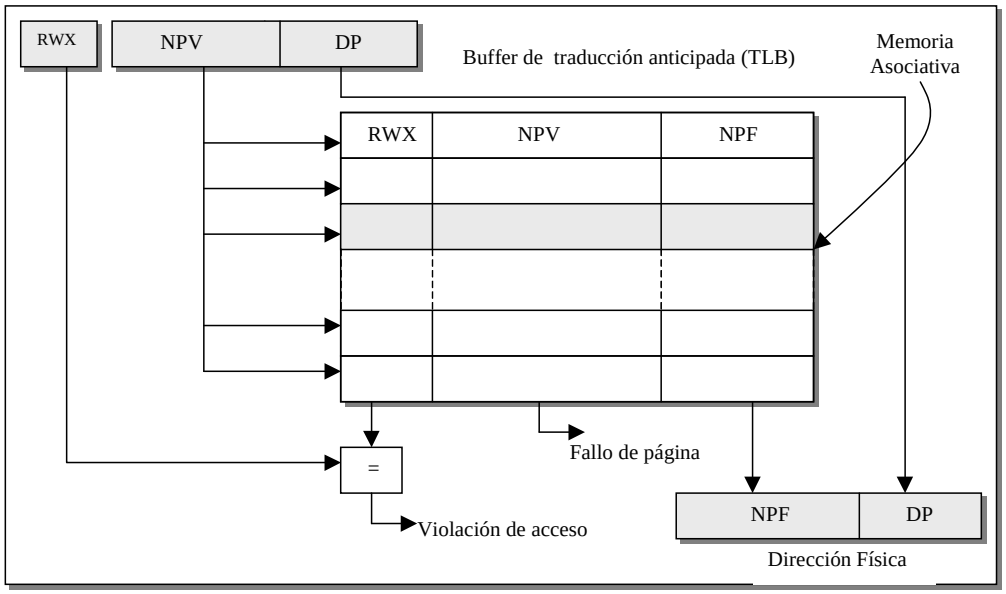


2.1.3. Buffer de traducción anticipada (TLB)

En principio toda referencia a memoria virtual requiere dos accesos a la memoria física: uno para acceder al elemento de la TP, y otro para acceder a la memoria física. Por tanto, un esquema de memoria virtual como el que acabamos de estudiar duplicaría el tiempo de acceso a memoria.

Para evitar este inconveniente los esquemas de memoria virtual utilizan una cache especial para los elementos de la TP, llamada usualmente *buffer de traducción anticipada* (TLB, *Translation Lookaside Buffer*).

El TLB funciona lo mismo que una memoria cache, y contiene aquellas entradas de la TP a las que se han accedido recientemente. Por el principio de localidad temporal, la mayoría de las referencias a memoria corresponderán a posiciones incluidas en páginas recientemente utilizadas. Por ese motivo, la mayoría de las referencias involucran a entradas de la TP presentes en el TLB. Normalmente el TLB utiliza una correspondencia totalmente asociativa, por lo que una entrada de la TP puede ubicarse en cualquier posición del TLB



2.2. Interacción entre la memoria virtual y la memoria cache

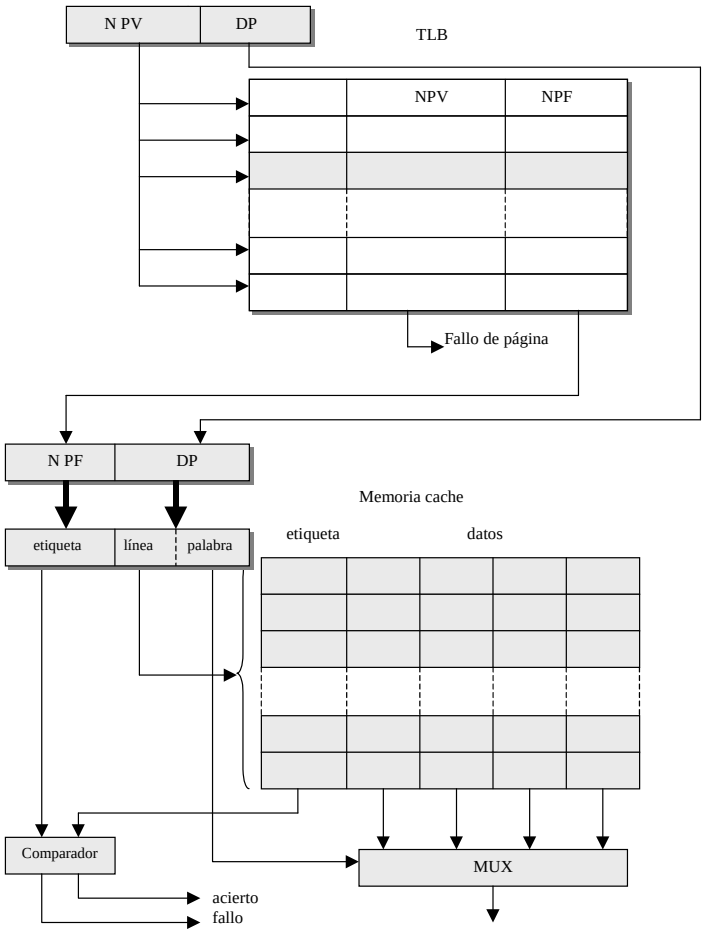
La memoria virtual debe interactuar con la memoria cache (no con la cache que implementa el TLB, sino con la caché de la memoria principal).

Para traducir una DV el sistema de memoria consulta el TLB para comprobar si la entrada de la TP se encuentra en él. Si es así, se genera la dirección real (física), combinando el NPF con el DP. Si no, se accede a la TP en busca del elemento correspondiente. Una vez que se ha generado la dirección física, compuesta por una etiqueta y un número de conjunto, se consulta la cache para ver si el bloque que contiene la palabra referenciada se encuentra en dicho conjunto. Si es así, se envía al procesador. Si no, se produce un fallo de cache y se busca la palabra en memoria principal.

Por tanto, la dirección virtual debe pasar primero por el TLB antes de que la dirección física pueda acceder a la cache, lo que alarga el tiempo de acierto. Este mecanismo se puede acelerar utilizando dos alternativas: a) Acceder en paralelo (simultáneamente) al TLB para buscar el NPF y al directorio de la cache para buscar el bloque, y b) Utilizar caches con direcciones virtuales

2.2.1. Acceso paralelo a TLB y caché

La primera alternativa requiere que la longitud del campo de desplazamiento DP de la DV (igual al DP de la DF) sea mayor o igual que los campos de *conjunto* y *palabra* del formato de la DF para la cache, tal como se muestra en la siguiente figura. De esta forma será posible realizar en paralelo la búsqueda en el TLB del NPF que se corresponde con la etiqueta, y en el directorio de la cache el nº de bloque (línea), que junto a la palabra dentro del bloque, se corresponde con el DP, del que se dispone desde el instante que se genera la DV, pues no requiere traducción.



Esto significa que el tamaño de la caché viene impuesto por el tamaño de la página y el grado de asociatividad. Por ejemplo, si tenemos un tamaño de página de 1K, el DP tendrá 10 bits. Si el número de palabras por bloque es 8 (3 bits de palabra) quedarán 7 bits para el campo de conjunto de la dirección física de la cache. Si la asociatividad fuese 1 (correspondencia directa) con esos 7 bits podríamos distinguir tan sólo entre 128 marcos de bloque o líneas de la caché, lo que impondría un tamaño de caché de 128 x 8 = 1024 = 1K.

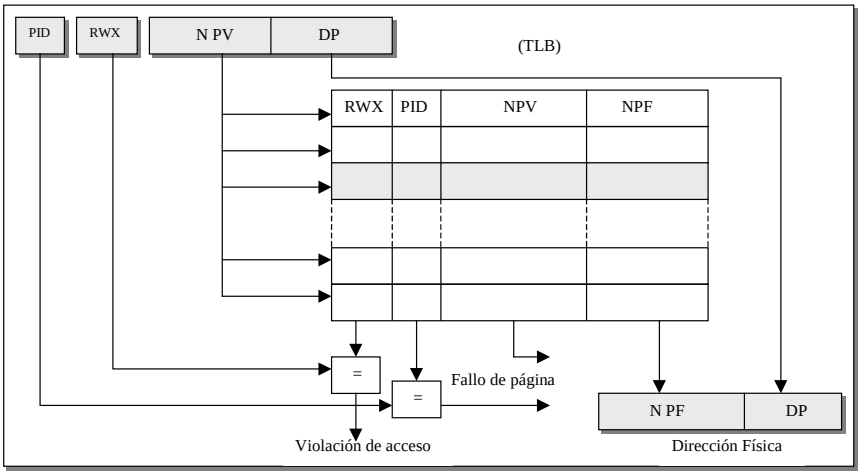
Para aumentar este tamaño tendríamos que aumentar la asociatividad. Por ejemplo, si el grado de asociatividad es 2, el tamaño de la caché sería de 2K. Cuando aumenta el grado de asociatividad, también aumentan las comparaciones que debemos realizar con la etiqueta obtenida del TLB, ya que se deben comparar con ella todas las etiquetas del conjunto de la cache (tantas como vías)

2.2.2. Caches con direcciones virtuales

La segunda alternativa de utilizar caches direccionadas virtualmente tiene la ventaja de que en caso de acierto en la caché, no es necesario el proceso de traducción. Sin embargo, esta alternativa tiene dos problemas fundamentales. El primero surge en las conmutaciones de procesos. Cuando se cambia de proceso, la misma dirección virtual de cada proceso referencia diferentes direcciones físicas, lo que exige que se limpie la caché y esto supone una gran fuente de fallos forzosos. El segundo se origina porque es posible que distintos procesos con diferentes direcciones virtuales se correspondan con la misma dirección física (*aliasing*). Esto puede producir dos copias del mismo dato en una cache virtual, lo que ocasiona problemas de inconsistencia si hay escrituras.

2.3. Múltiples espacios virtuales

Una forma de acelerar la conmutación de procesos evitando tener que borrar la TP (que puede estar en TLB o memoria invertida) es extender las entradas con un campo que contiene el identificador de proceso (PID).



2.4. Políticas de búsqueda (fetch)

2.4.1. Prebúsqueda

Análoga a la utilizada en memoria cache)

2.4.2. Búsqueda por demanda

2.5. Políticas de sustitución (*replacement*)

2.5.1. Aleatoria

Elige una página aleatoriamente, sin mirar el número de referencias o el tiempo que la página lleva en memoria principal. En general esta política tiene unos resultados pobres, excepto en entornos donde existe poca localidad como en las bases de datos.

2.5.2. FIFO (First In First Out)

Se sustituye la página que lleva más tiempo residente en memoria. Utiliza una cola FIFO y hace un uso pobre de la localidad temporal

2.5.3. Reloj (FINUFO: *First In Not Used First Out*)

Es una mejora de la FIFO en la que también se chequea si una página ha sido referenciada, haciendo mejor uso de la localidad temporal. Para implementar esta política se mantiene una cola como en la FIFO, pero circular, con un puntero a la página candidata a ser sustituida, y un *flag de uso* asociado a cada página. El *flag de uso* se pone a 1 cuando la página es referenciada con posterioridad a su carga inicial en la memoria física. Al producirse un fallo de página se examina el *flag de uso* de la página señalada por el puntero de la FIFO. Si está a 0 la correspondiente página es sustituida, pero si vale 1 se borra el *flag de uso* (se pone a 0) y se avanza el puntero una posición, continuando este procedimiento hasta encontrar una página con el *flag de uso* a 0.

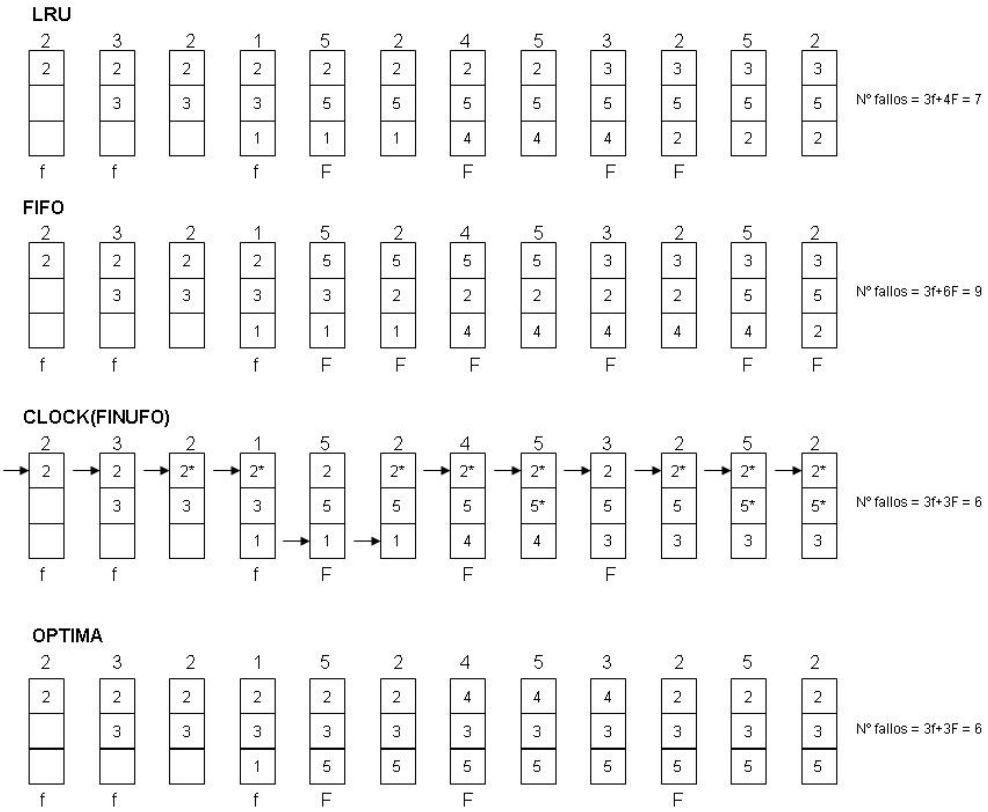
2.5.4. LRU (Least Recently Used)

Análoga a la utilizada en memoria caché

2.5.5. Optima (MIN)

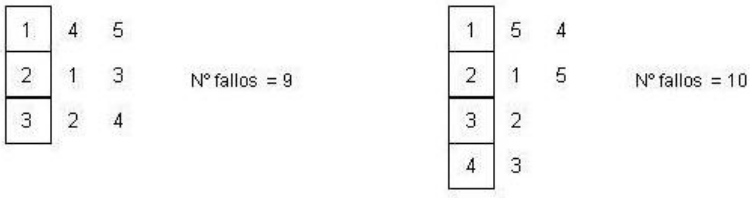
Se trata de la mejor política posible: sustituir la página que vaya a tardar más tiempo en ser referenciada en el futuro (Belady). Aunque esta política tiene el mínimo número posible de fallos de página (de aquí el nombre de política MIN), no se puede llevar a la práctica en tiempo real, y se utiliza como una referencia teórica para medir la eficiencia de otras políticas en entornos experimentales.

Veamos el comportamiento de cada una de las políticas frente al siguiente perfil de referencias a páginas: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2:



En el dibujo anterior, en la política FINUFO, la flecha representa el puntero a la página candidata a ser sustituida, y el asterisco el bit de *flag* a 1.

Anomalía de Belady. Comportamiento anómalo de la política LRU frente a determinados perfiles de referencia cuando, aumentando el tamaño de la memoria física, también aumenta el número de fallos, en contra de lo que cabría esperar. Por ejemplo, sobre 2 memorias física, una de 3 marcos de página y otra de 4 marcos de página, frente al perfil de referencias a página siguiente: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, se generan los siguientes fallos de página:



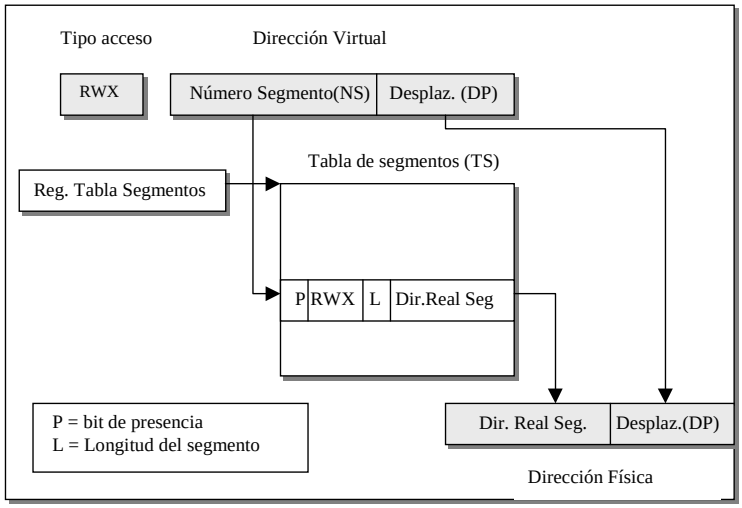
Como vemos, sobre la segunda memoria física (que es de mayor tamaño que la primera) se produce un fallo de página más que sobre la primera, para el mismo perfil de referencias. Este comportamiento es el que se denomina *anomalía Bealy*.

3. Memoria virtual segmentada

Mientras que la paginación es transparente para el programador, y le proporciona un espacio mayor de direcciones, la segmentación es normalmente visible para el programador, y proporciona una forma lógica de organizar los programas y los datos, y asociarle los privilegios y atributos de protección.

La segmentación permite que el programador vea la memoria constituida por múltiples espacios de direcciones o segmentos. Los segmentos tienen un tamaño variable, dinámico.

Usualmente, el programador o el sistema operativo asignará programas y datos a segmentos distintos. Puede haber segmentos de programa distintos para varios tipos de programas, y también distintos segmentos de datos. Se pueden asignar a cada segmento derechos de acceso y uso. Las direcciones virtuales estarán constituidas en este caso por un número de segmento (NS) y un desplazamiento dentro del segmento (DP). El proceso de traducción de dirección virtual a física es análogo al de la memoria virtual paginada, con la diferencia que ahora tenemos una tabla de segmentos (TS) cuyas entradas (denominadas también descriptores de segmento) contienen, además de los bits de control y la dirección real del segmento, la longitud L del mismo, ya que los segmentos tienen longitud variable. En la siguiente figura hemos representado esquemáticamente el proceso de traducción.



Esta organización tiene ciertas ventajas para el programador, frente a un espacio de direcciones no segmentado:

- Simplifica la gestión de estructuras variables de datos. Si el programador no conoce a priori el tamaño que puede llegar a tener una estructura de datos particular, no es necesario que lo presuponga. A la estructura de datos se le asigna su propio segmento, y el sistema operativo lo expandirá o lo reducirá según sea necesario.
- Permite modificar los programas y recompilarlos independientemente, sin que sea necesario volver a enlazar y cargar el conjunto entero de programas. De nuevo, esto se consigue utilizando varios segmentos.
- Permite que varios procesos compartan segmentos. Un programador puede situar un programa correspondiente a una utilidad o una tabla de datos de interés en un segmento, que puede ser direccionado por otros procesos.
- Se facilita la protección. Puesto que un segmento se construye para contener un conjunto de programas o datos bien definido, el programador o el administrador del sistema puede asignar privilegios de acceso de forma adecuada.

3.1. Políticas de ubicación (*placement*) para memorias segmentadas

Se identifican los huecos de memoria principal por su tamaño (longitud) y dirección inicial, y se reúnen en una lista. La política de ubicación determinará la ordenación previa de la lista. Una vez que se decide el hueco donde se ubica el segmento, se actualiza la lista de huecos con el que se acaba de crear (a no ser que el segmento mida exactamente igual que el hueco de memoria utilizado). Si no se puede encontrar un hueco apropiado para el segmento, interviene la política de sustitución. Las tres políticas de ubicación de segmentos más utilizadas son las siguientes:

3.1.1. Mejor ajuste (*best fit*)

La lista de huecos se mantiene ordenada en orden creciente de tamaño y se ubica el segmento en el primer hueco con capacidad suficiente para albergarlo.

3.1.2. Peor ajuste (*worst fit*)

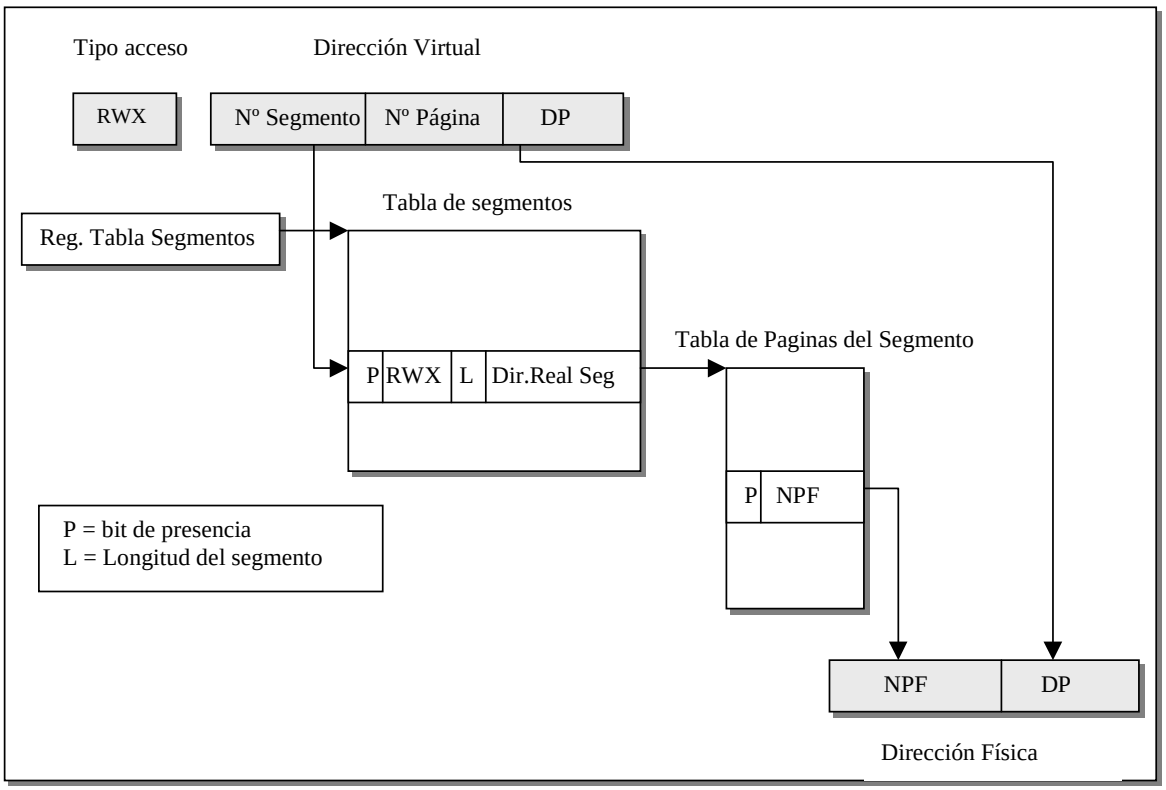
La lista de huecos se mantiene ordenada en orden decreciente de tamaño y se ubica el segmento en el primer hueco con capacidad suficiente para albergarlo.

3.1.3. Primer ajuste (*first fit*)

La lista de huecos se mantiene ordenada en orden creciente de las direcciones iniciales de los huecos, y se ubica el segmento en el primer hueco con capacidad suficiente para albergarlo. Con esta política, cuando transcurre un cierto tiempo, se acumulan un número elevado de huecos pequeños próximos a la cabeza de la lista, penalizando las búsquedas. Esto se puede evitar adelantando cíclicamente, después de cada búsqueda un hueco la posición inicial de la lista.

4. Memoria con segmentos paginados

Como vimos en el apartado anterior, la segmentación presenta una serie de propiedades ventajosas para el programador, sin embargo, la paginación proporciona una forma más eficiente de gestionar el espacio de memoria. Para combinar las ventajas de ambas, algunos sistemas permiten una combinación de ambas, es decir, un sistema virtual con segmentos paginados. El mecanismo de traducción de DVs a DFs no es más que la composición del mecanismo de la memoria segmentada y el de la paginada, tal como se muestra en la siguiente figura:



5. Ejemplo de sistema de memoria virtual: procesador Pentium II

El Pentium II dispone de un sistema de gestión de memoria virtual con posibilidad de segmentación y paginación. Los dos mecanismos se pueden activar o desactivar con independencia, dando pues lugar a cuatro formas de funcionamiento del sistema de memoria:

1) Memoria no segmentada no paginada: la dirección virtual coincide con la dirección física. Esta alternativa resulta útil cuando el procesador se utiliza como controlador de sistemas empotrados.

2) Memoria paginada no segmentada: la memoria constituye un espacio lineal de direcciones paginado. La protección y la gestión de memoria se realizan a través de la paginación.

3) Memoria segmentada no paginada: la memoria constituye un conjunto de espacios de direcciones virtuales (lógicas). Esta alternativa presenta la ventaja frente a la paginación en que proporciona, si es necesario, mecanismos de protección a nivel de byte. Además, garantiza que la tabla de segmentos se encuentra ubicada en el procesador cuando el segmento está en memoria. Por ello, la segmentación sin páginas da lugar a tiempos de acceso predecibles.

4) Memoria segmentada paginada: se utilizan simultáneamente los dos mecanismos, la segmentación para definir particiones lógicas de memoria en el control de acceso, y la paginación u para gestionar la asignación de memoria dentro de las particiones.

5.1. Mecanismo de segmentación

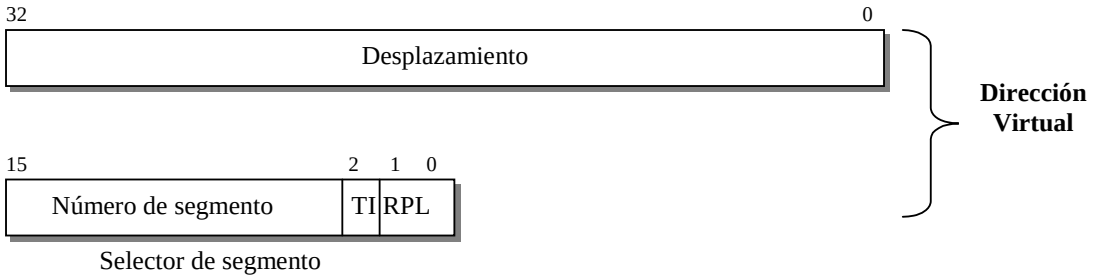
Cuando se activa la segmentación una *dirección virtual* (llamada *dirección lógica* en la terminología de Intel) consta de una referencia al segmento de 16 bits y un desplazamiento de 32 bits. Dos bits de la referencia al segmento se utilizan para el mecanismo de protección, y los 14 bits restantes para especificar al segmento en cuestión (16K segmentos). Con una memoria segmentada, el espacio de memoria virtual total que ve el usuario es 2^{46} = 64 Terabytes (TBytes). El espacio de direcciones físicas utiliza direcciones de 32 bits, con una capacidad máxima de 4 GBytes.

Existen dos formas de protección asociadas a cada segmento: *el nivel de privilegio* y *el atributo de acceso*.

Hay cuatro niveles de privilegio, desde el más protegido (nivel 0), al menos protegido (nivel 3). *El nivel de privilegio* asociado a un segmento de datos constituye su *clasificación*; mientras que el nivel de privilegio asociado a un segmento de programa constituye su *acreditación* (*clearance*). Un programa en ejecución puede acceder a un segmento de datos sólo si su nivel de acreditación es menor (tiene mayor privilegio) o igual (igual privilegio) que el nivel de privilegio del segmento de datos.

La forma de utilizar los niveles de privilegio depende del diseño del sistema operativo. Lo usual es que el nivel de privilegio 1 lo utilice la mayor parte del sistema operativo, y el nivel 0 una pequeña parte del mismo, la dedicada a la gestión de memoria, la protección y el control del acceso. Esto deja dos niveles para las aplicaciones. En muchos sistemas, las aplicaciones utilizan el nivel 3, dejándose sin utilizar el nivel 2. Los aplicaciones que implementan sus propios mecanismos de seguridad, como los sistemas de gestión de bases de datos, suelen utilizar el nivel 2. Además de regular el acceso a los segmentos de datos, el mecanismo de privilegio limita el uso de ciertas instrucciones. Por ejemplo, las instrucciones que utilizan los registros de gestión de memoria, sólo pueden ejecutarse desde el nivel 0; y las instrucciones de E/S sólo pueden ejecutarse en el nivel determinado por el sistema operativo, que suele ser el nivel 1. *El atributo de acceso* al segmento de datos especifica si se permiten accesos de lectura/ escritura o sólo de lectura. Para los segmentos de programa, el atributo de acceso especifica si se trata de acceso de lectura/ejecución o de sólo lectura.

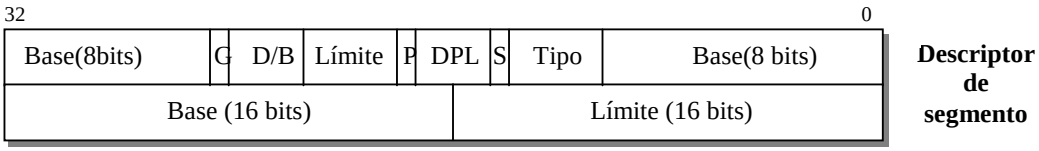
El mecanismo de traducción de dirección para la segmentación hace corresponder una *dirección virtual* con una *dirección lineal*. Una dirección virtual consta de un desplazamiento de 32 bits y un selector de segmento de 16 bits.



El selector de segmentos consta de los siguientes campos:

- TI: Indica si la traducción va a utilizar la tabla de segmento global o local.
- Número de segmento: se utiliza como índice en la tabla de segmentos.
- RPL: Indica el nivel de privilegio del acceso.

Cada elemento de la tabla de segmentos consta de 64 bits, como muestra la siguiente figura:



Los campos son los siguientes:

- Base (32 bits): dirección de comienzo del segmento dentro del espacio lineal de direcciones
- Límite (20 bits): tamaño del segmento
- Nivel de privilegio(DPL) (2 bits)
- Bit de segmento presente(P) (1 bit): indica si el segmento está disponible en memoria principal
- Bit S (1 bit): determina si el segmento es del sistema
- Tipo (4 bits) determina el tipo de segmento e indica los atributos de acceso
- Bit de granularidad (G) (1 bit): indica si el campo límite se interpreta en unidades Bytes o 4KBytes
- Bit D/B (1 bit): indica si los operandos y modos de direccionamiento son de 16 ó 32 bits (en segmentos de código)

5.2. Mecanismo de paginación

Cuando se activa la paginación los programas utilizan directamente direcciones lineales. El mecanismo de paginación del Pentium II utiliza una tabla de páginas de dos niveles. El primero es un directorio de páginas, que contiene hasta 1.024 elementos. Esto divide los 4 GBytes del espacio lineal de memoria en 1.024 grupos de páginas, cada grupo con 4 Mbytes de capacidad y con su propia tabla de páginas. La flexibilidad del sistema de gestión de memoria permite utilizar un directorio de páginas para todos los procesos, un directorio de páginas para cada proceso o una combinación de ambos. El directorio de páginas del proceso en curso está siempre en memoria principal. Las tablas de páginas pueden estar en memoria virtual.

La siguiente figura ilustra la combinación de los mecanismos de segmentación y paginación en la que no aparece el TLB ni la memoria caché. El Pentium II permite seleccionar dos tamaños de páginas de 4 KBytes ó de 4 MBytes. Cuando se utilizan páginas de 4 MBytes, hay un sólo nivel

en la tabla de páginas. El uso de páginas de 4 MBytes reduce las necesidades de memoria. Con páginas de 4 KBytes, una memoria principal de 4 GBytes necesita del orden de 4 MBytes de memoria sólo para la tabla de páginas. Con páginas de 4 MBytes, una única tabla, de 4 KBytes de longitud, es suficiente para la gestión de las páginas.