Unidad 5: MEMORIA

Gestión de memoria. Protección. Técnicas de intercambio. Reubicación. Asignación de particiones múltiples. Registros. Memoria real. Fragmentación. Particiones de tamaño fijo. Particiones de tamaño variable. Compactación. Memoria Virtual. Modo de funcionamiento. Paginación- segmentación.

MEMORIA

TÉRMINOS UTILIZADOS EN MEMORIA

PUNTO DE MEMORIA

Las memorias utilizan el almacenamiento binario, es decir que la información elemental almacenada es el bit, y al **soporte físico** del bit, se lo denomina **punto de memoria**. Son ejemplos clásicos: un **biestable** (Flip-Flop), **capacitor**, **burbuja magnética**, **núcleo de ferrite**.

BYTE

Es una unidad de almacenamiento equivalente a **8 bits**. Su nombre proviene de **B**inar**y Te**rmin (Término Binario).

CARÁCTER

Es la cantidad de bits necesarios para representar un símbolo de un código alfabético, numérico o especial.

PALABRA O WORD

Es una unidad lógica de información que puede ser manipulada por el procesador en un instante determinado. Generalmente se encuentra conformada por un número exacto de bytes.

VOLATILIDAD

Es la **característica** que tienen las memorias de **alterar** o **perder** su contenido cuando se produce una falla en el suministro eléctrico o se produce el corte del mismo.

DIRECCIÓN DE MEMORIA

Es la ubicación física de una palabra en memoria.

DIRECCIONAMIENTO

Es la función desarrollada por el procesador para localizar un dato o una instrucción dentro de la memoria.

TIEMPO DE ACCESO

Es el tiempo que transcurre entre el instante en que se lanza una operación de lectura en memoria y el instante en que se dispone de la primera información buscada.

CAPACIDAD DE MEMORIA

Se denomina así a la cantidad máxima de información que puede contener una memoria. En la actualidad estas capacidades suelen medirse en Mbytes. Y Gbytes.

CAUDAL O VELOCIDAD DE TRANSFERENCIA

Se denomina así a la cantidad de información que puede leerse o escribirse en una unidad de tiempo. Normalmente se la mide en bytes o múltiplos por segundos.

OPERACIONES BÁSICAS DE MEMORIA

ESCRITURA

Es la operación durante la cual **se registran nuevos datos** en la celda de memoria. La operación de escritura **es destructiva**, ya que el nuevo contenido reemplaza al anterior. Además, dependiendo de la tecnología utilizada, la escritura **puede requerir** del **borrado previo** de la celda, este es el caso de las **memorias de núcleos**.

LECTURA

Es la operación mediante la cual se **extrae** la **información contenida** en la celda. En el caso de la lectura, la destrucción o no de la información contenida también depende de la tecnología utilizada. Las memorias que utilizan **semiconductores no son destructivas**, mientras que las que utilizan una **tecnología de núcleo sí lo son**.

MODOS DE ACCESO

Básicamente existen dos formas de acceder a una determinada palabra almacenada:

ACCESO DIRECTO O ACCESO ALEATORIO

Este es el caso de las memorias formadas por núcleos o semiconductores donde cualquier celda puede ser accedida en un momento determinado.

ACCESO SECUENCIAL:

En este caso para acceder a un dato determinado, es necesario esperar que pasen ante el dispositivo todos los elementos previos. Un ejemplo de este tipo de acceso son las cintas magnéticas, donde para acceder a un determinado registro es necesario pasar por encima de todos aquellos que están antes que él.

MODOS DE DIRECCIONAMIENTO

Se denominan **modos de direccionamiento** o **técnicas de acceso a la Memoria Principal** a los métodos utilizados para seleccionar la ***dirección*** en la que se quiere extraer o almacenar un dato.

Estas técnicas están relacionadas tanto con la arquitectura del hardware del ordenador como con el software utilizado.

Antes de entrar al estudio detallado de cada uno de los modos de direccionamiento, debemos tener presente que:

- Hay tareas de proceso y programas que utilizan varios modos de direccionamiento.
- Cada modo puede combinarse con uno o varios de los otros modos ya conocidos.

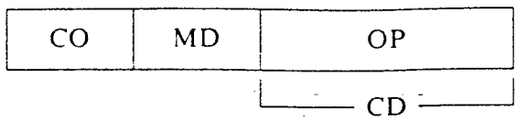
Las Unidades Centrales de Proceso de las arquitecturas de los ordenadores que se utilizan hoy en día permiten utilizar varios modos de direccionamiento. Veamos primero cada uno por separado para luego realizar un estudio de los modos combinados más utilizados.

DIRECCIONAMIENTO INMEDIATO

En él la instrucción contiene ya el operando y no la dirección en que se encuentra. Dependiendo del tipo de formato de instrucción, este direccionamiento se utiliza de dos formas diferentes:

A) CON FORMATO DE INSTRUCCIÓN DE LONGITUD FIJA

En estos casos la longitud de la instrucción coincide con la longitud de la palabra de memoria; es decir, la instrucción completa, con operando incluido, ocupa una posición de memoria. El formato de instrucción contiene los siguientes campos:



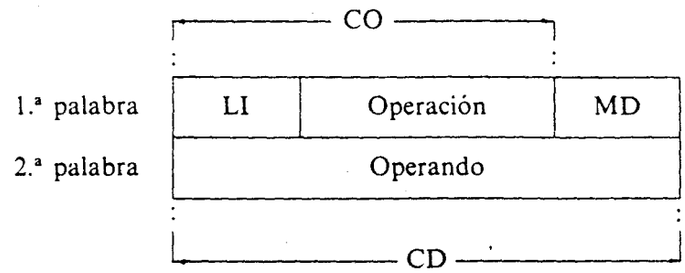
CO: código de operación.
MD: modo de direccionamiento.
CD: campo de dirección (operando).

Este modo de direccionamiento es el más rápido de procesar, ya que para acceder al operando necesita sólo el acceso a memoria en búsqueda de la instrucción.

El número de bits del operando queda limitado por la longitud del CD. Combinándolo con otros MD, se utiliza en minis y grandes ordenadores. Una aplicación habitual que utiliza este MD es la de realizar operaciones aritméticas rápidas con pequeños valores de operando (ejemplo: SUMAR 10).

B) CON FORMATO DE INSTRUCCIÓN DE LONGITUD VARIABLE

En la mayoría de los ordenadores fabricados con microprocesadores actuales se aprecia que la longitud de su Juego de Instrucciones no es constante. Según el tipo de operación, la longitud de la instrucción es variable. En estos casos la instrucción está compuesta por varias palabras ubicadas en posiciones contiguas de la memoria. En el caso particular de una instrucción de dos palabras de longitud, *el* formato *tipo de* instrucción con este MD contiene los siguientes campos:



El campo de operación CO se divide en dos subcampos: el del código de operación propiamente dicho y un segundo campo de un número reducido de bits, cuya interpretación indica, al circuito de control, la longitud de la instrucción (LI).

El campo de dirección, en este caso, lo forma la segunda palabra de la instrucción que contiene el operando.

Las características principales de este MD son las siguientes:

- Se requiere un ciclo con referencia a memoria para leer el dato contenido en la 2ª palabra de la instrucción.

- Una aplicación habitual que utiliza este MD es la de realizar operaciones aritméticas con valores de operando grandes.

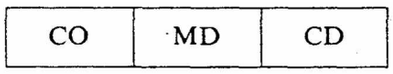
Se utiliza en micros, minis y grandes ordenadores.

DIRECCIONAMIENTO DIRECTO

Se diferencia del anterior en que el CD de la instrucción no contiene al operando, sino a su dirección final o efectiva.

Se entiende por dirección final (DF) a la dirección en que se encuentra el dato al que se quiere acceder en la memoria, en un proceso de lectura.

El formato tipo de instrucción con este direccionamiento contiene los siguientes campos:



Las características principales de este direccionamiento son las siguientes:

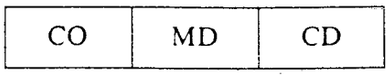
- No son necesarios cálculos previos para determinar la dirección final.
- Necesita un ciclo de memoria exclusivamente para acceder al operando, que ahora puede estar almacenado en cualquier posición de la memoria usuario. . Si se utilizara este direccionamiento con carácter exclusivo, el CD debería tener una longitud suficientemente grande como para poder acceder a todo el mapa de la memoria.
- Con formatos de instrucción de dos palabras este MD posibilita una mayor capacidad de direccionamiento.

DIRECCIONAMIENTO INDIRECTO

En este direccionamiento, conocido también por **indirección**, el contenido del CD, a diferencia del modo directo, apunta a la dirección de una palabra de memoria, **dirección intermedia**, cuyo contenido será interpretado como la dirección (DF), lo que implica otro ciclo más de memoria para acceder al operando.

No son necesarios cálculos previos para determinar la dirección en la que se encuentra el operando.

El formato tipo de instrucción con este direccionamiento contiene los siguientes campos:



Las características principales de este direccionamiento son las siguientes:

- No son necesarios cálculos previos para determinar la dirección final.
- Necesita dos ciclos de memoria para acceder al operando; durante el primero se accede a la dirección intermedia y en el segundo se accede al operando.
- Con este caso se incrementa notablemente la capacidad de direccionamiento al poder utilizar todos los bíts de la palabra de memoria.
- Si la instrucción fuera de salto, la DF sería transferida simultáneamente a los registros CP (contador de programa) y D (registro de direcciones), para de esta forma ganar tiempo e iniciar la búsqueda de la siguiente instrucción.

El direccionamiento indirecto se utiliza en muchas aplicaciones que hacen uso de datos alojados en posiciones distantes en la memoria.

La indirección es un método eficaz para transferir los parámetros de un programa principal a subrutinas.

El montaje de enlaces en bancos de datos se realiza mediante este direccionamiento de una forma sencilla.

Este direccionamiento permite con una sola instrucción, de un programa, acceder a distintas zonas de la memoria.

Todos los ordenadores que utilizan memoria paginada necesitan de un direccionamiento indirecto.

DIRECCIONAMIENTO RELATIVO

Este direccionamiento se caracteriza porque el valor numérico de la **dirección final** de la palabra a acceder se consigue tratando conjuntamente la información contenida en el CD del RI y la contenida en algún registro determinado.

La característica fundamental del direccionamiento relativo es que posibilita el acceso a un conjunto de direcciones determinadas, a partir de una considerada como dirección de referencia.

Existen varios modos de direccionamiento relativo, según donde se encuentre almacenada la dirección de referencia o cuál sea el proceso para determinar la dirección final del dato.

Los ordenadores que tienen algunas instrucciones, cuya longitud del CD es insuficiente para direccionar la totalidad del mapa de la memoria, van provistos en su diseño de algún direccionamiento relativo a otro registro distinto del RI.

En algunos casos la relatividad se consigue obteniendo la dirección final mediante cálculos aritméticos sencillos, habitualmente sumas o restas, cuyos resultados se transfieren al registro D como dirección de acceso a la memoria.

Otros métodos transfieren simultáneamente la información al registro D combinándola por campos de bits.

Las posibilidades del direccionamiento relativo pueden ser muy numerosas. En los apartados siguientes se describen algunos de estos direccionamientos.

DIRECCIONAMIENTO RELATIVO AL REGISTRO CONTADOR DE PROGRAMA

El programador que haga uso en exclusiva de este direccionamiento debe tener en cuenta la proximidad entre instrucciones y datos.

La dirección final se determina por suma o resta al contenido del registro CP, dirección base o relativa, de un valor de desplazamiento contenido en el CD de la instrucción en curso, almacenada en el RI.

Este direccionamiento posibilita un *rango de memoria direccionable*, a partir de la dirección de referencia.

DIRECCIONAMIENTO DE LA MEMORIA CON REGISTROS ADICIONALES

Se entiende que los direccionamientos anteriormente descritos son suficientes para arquitecturas de ordenadores sencillos; pero existen otros más potentes que, utilizando unidades operadoras complementarias, contribuyen a mejorar notablemente el rendimiento del ordenador, transfiriendo la información que procesan no exclusivamente a un registro único, sino hacia alguno de los que componen una batería de Registros de Propósito General (RPG), o Registros de Trabajo, que pueden utilizarse indistintamente, ya sea como registro destino o registro fuente.

Los ordenadores en cuya arquitectura se dispone de este tipo de registros posibilitan modernas técnicas de direccionamiento que ofrecen notables ventajas al programador, además de mejorar la eficiencia de la Unidad Central de Proceso al reducir los tiempos de espera en los accesos a memoria.

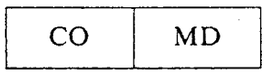
La inserción en la arquitectura de una batería de registros requiere una pequeña circuitería adicional y, por tanto, un pequeño incremento del coste del hardware del equipo.

A continuación vamos a ver cómo trabajan estos registros al describir nuevas técnicas de direccionamiento.

DIRECCIONAMIENTO MEDIANTE REGISTRO

En este direccionamiento el operando está contenido en uno de los registros generales, designado como Registro de Trabajo (RT), de acceso muy rápido.

El formato de instrucción contiene los siguientes campos:



Un primer campo, que contiene el código de operación CO y un segundo campo, identificado por MD de un número reducido de bits, cuya interpretación le indica al circuito de control en qué registro está contenido el operando, previamente almacenado en RT.

Las características principales de este direccionamiento son las siguientes:

Requiere un sólo ciclo con referencia a memoria, el de la instrucción, para acceder al operando.

Una aplicación habitual que utiliza este direccionamiento es la de realizar operaciones aritméticas con valores de operando suficientemente grandes, pero que no desborden la capacidad de un registro.

Este direccionamiento se implementa en ordenadores con formato de instrucción de longitud variable.

Se utiliza en micros, minis y grandes ordenadores.

DIRECCIONAMIENTO INDEXADO

En el direccionamiento indexado la dirección relativa es el contenido de un registro que llamaremos registro índice (RIN), cuya longitud es idéntica a la del registro D, y que contiene un número entero, generalmente positivo, llamado índice. La lógica de este registro le permite realizar, además de las operaciones de lectura y escritura, incrementos y decrementos de forma automática, de una magnitud limitada, ejemplo: de 4 en 4.

El utilizar este RIN como soporte de almacenamiento temporal de un índice, que realiza funciones de puntero, facilita la rápida exploración secuencial de los elementos de un vector de datos, tanto en forma ascendente como descendente, permitiendo al programador que con una sola instrucción se pueda direccionar cualquier elemento de dicho vector.

Si este direccionamiento no se combina con otras técnicas de acceso a la memoria, el procesamiento de una instrucción implica las siguientes fases:

- Se toma como dirección del primer elemento del vector de datos al valor contenido en el CD de la instrucción.
- El orden en que se encuentra el elemento, dentro del 'vector de datos, viene determinado por el contenido actualizado del RIN.
- La dirección final se determina por la suma de los contenidos en el CD y en el RIN, transfiriendo el resultado al registro D.

DIRECCIONAMIENTO RELATIVO A REGISTRO BASE

Por medio de este direccionamiento la dirección final se obtiene sumando, al contenido del Registro de Dirección Base (RDB), el contenido del CD de la instrucción, que se interpreta aquí como un desplazamiento.

El RDB tiene la misma longitud que el registro D y presenta la característica de que el hardware del ordenador lo hace no accesible al programador, al igual que otro reducido número de registros.

Cuando un programador diseña un programa en lenguaje de alto nivel, desconoce las posiciones en que el compilador va a situar a las instrucciones y operandos en la memoria, aunque el sistema operativo haya reservado zonas de memoria para los programas de usuario.

Este direccionamiento permite a los Sistemas Operativos multiusuario reubicar fácilmente a los programas de usuario sin solaparse en las mismas posiciones de memoria, al tomar como referencia una dirección que denominamos dirección base (DB).

DIRECCIONAMIENTO POR COMBINACIÓN DE VARIOS MODOS

En lo expuesto hasta ahora se ha considerado que se trabajaba con cualquier direccionamiento en exclusiva. En realidad, esto no ocurre en los ordenadores que se manejan. Normalmente la dirección final se determina combinando varios modos de direccionamiento.

Las posibles combinaciones de dos o más direccionamientos conllevan siempre un orden, generalmente preestablecido en función de la arquitectura del ordenador, para el tratamiento de los diferentes direccionamientos especificados en la instrucción.

Las prioridades del orden y las técnicas con que son tratadas las diferentes combinaciones de direccionamiento dan lugar a nuevos modos, de los cuales los más usados son:

COMBINACIÓN DEL MODO MEDIANTE REGISTRO

El modo combinado consiste en interpretar primero la dirección del RPG, en cuyo contenido se encuentra la dirección final:

$$DF = (RPG)$$

COMBINACIÓN DEL MODO INDIRECTO Y EL MODO INDEXADO

Cuando en la ejecución de una instrucción primero se interpreta la indirección y luego la indexación, el nuevo direccionamiento se conoce por post-indexación. Este caso queda resumido en la forma de determinar la dirección final:

$$DF = ((CD)) + (RIN)$$

Una aplicación habitual que utiliza este modo, es la de realizar exploraciones a una tabla de datos, con llamada a una sola instrucción.

La otra combinación, menos utilizada, de interpretar primero la indexación y después la indirección se conoce por pre-indexación.

La forma de determinar la dirección final se resume en la siguiente expresión:

$$DF = ((CD) + (RIN))$$

Los ordenadores cuyas arquitecturas posibilitan estos modos referenciados, tienen un bit en el campo de MD, cuyo contenido indica a la unidad de control el orden a seguir en la ejecución.

COMBINACIÓN DEL MODO RELATIVO A BASE E INDIRECTO

Este otro modo combina los dos modos referenciados. La *dirección final* se determina de la siguiente forma: en primer lugar, se realiza la suma de los contenidos del RB y el CD, y posteriormente se resuelve la indirección.

Este caso queda resumido en la forma de determinar la dirección Final:

$$DF = ((RB) + (CD))$$

Este direccionamiento resulta muy útil cuando se desea acceder a múltiples direcciones de comienzo de varias zonas de datos.

A continuación se trata el modo combinado, que parte de este que se acaba de describir.

COMBINACIÓN DEL MODO RELATIVO A BASE, INDIRECTO E INDEXADO CON PRE Y POST INDEXACIÓN

Este nuevo direccionamiento, combinación de los tres referenciados, permite que con una sola instrucción se tenga acceso a un dato situado en una posición de memoria de una de las tablas agrupadas en una zona contigua de la memoria, cuya dirección de comienzo está localizada en la llamada tabla de direcciones bases.

La *dirección final* se determina por varias fases:

Se realiza la *preindexación* con la suma de los contenidos del RB y el. CD, este último como desplazamiento dentro de la tabla de direcciones bases (RB) + (CD), obteniéndose la 1.a dirección intermedia (1.a DI).

A ésta (1ª DI) le aplicamos la indirección ((RB) + (CD)) y nos traslada a la dirección de comienzo de la tabla de datos (2ª DI), donde se encuentra el operando.

Por último, se realiza la *post-indexación* sumando a ésta (2ª DI) el valor contenido en el (RPG), que realiza funciones de (RIN), y de esta forma se consigue la (DF) para acceder al operando.

Este caso queda resumido en la forma de determinar la dirección final:

$$DF = ((RB) + (CD)) + (RIN)$$

Como se ha podido comprobar, este direccionamiento resulta de suma utilidad para explorar zonas de la memoria con tablas de datos en posiciones contiguas, estando el conjunto de tablas ubicado en posiciones dispersas de la memoria.

Una aplicación habitual que utiliza este direccionamiento es la de realizar exploraciones de tablas de datos, ubicadas en memoria en posiciones dispersas.

Aunque se han presentado los modos combinados más utilizados, todavía quedan múltiples combinaciones realizables, que dependiendo de la localización de los datos facilitan la exploración en la memoria con la máxima eficiencia.

NIVELES JERÁRQUICOS DE MEMORIA

Es común que a la memoria de la computadora se la denomine **Memoria Central** para diferenciarla de otro tipo de almacenamiento como los discos, las cintas, etc.

Podemos considerar a la memoria como a un **conjunto de celdas** (casilleros) donde es posible almacenar datos o instrucciones. Estas celdas están numeradas, y la **unidad de control** (**UC**) es capaz de identificar a cada una por ese número o **dirección**.

Además de identificar a las celdas, la UC puede leer el contenido de una de ellas o reemplazarlo, escribiendo un nuevo dato o una nueva instrucción.

En una máquina lo ideal sería disponer de una memoria central muy rápida y de inmensa capacidad. Esta solución sería demasiado costosa e irrealizable técnicamente. Por este motivo es que se establecen jerarquías de memorias.

Estas jerarquías se dan básicamente a dos niveles, una **memoria central** relativamente **rápida pero pequeña**, y una **memoria auxiliar** de **gran capacidad pero más lentas** en cuanto los accesos.

De hecho pueden distinguirse **cinco niveles de memoria**, en cuanto a su utilización y localización en la arquitectura de la máquina:

MEMORIA CENTRAL

MEMORIAS DE REGISTROS

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 (banderas o registro de estado), registros y contadores, entran en esta categoría.

MEMORIA PRINCIPAL

Es aquella 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.

CLASIFICACIÓN DE LA MEMORIA PRINCIPAL SEGÚN SU TECNOLOGÍA DE FABRICACIÓN

NÚCLEOS MAGNÉTICOS

La describiremos sólo como curiosidad histórica, debido a que actualmente fueron reemplazados por los circuitos integrados (C.I.) o CHIPS en las de tipo: RAM y ROM.

El núcleo: la unidad elemental del sistema es un pequeño anillo o núcleo de material magnético que se magnetiza en uno u otro sentido, constituyendo así un dispositivo biestable, capaz de almacenar un bit de información.

SEMICONDUCTORES O CIRCUITOS INTEGRADOS

Son aquellos que utilizan el biestable electrónico o Flip-Flop como elemento de memoria. Un biestable es un sistema que no puede presentar más que dos estados lógicos diferentes que, en ausencia de estímulos externos, no tienen porque evolucionar, razón por la cual este dispositivo puede servir como elemento de memoria.

Cada bit está constituido por un biestable y sus circuitos asociados de lectura y escritura. Una diferencia fundamental que presenta esta tecnología es que la lectura no es destructiva y la escritura no exige puesta a cero previa como los núcleos magnéticos.

El tiempo de acceso de este tipo de memoria es de varias decenas de nanosegundos (10 a 120 ns., típicamente 70 ns).

CLASIFICACIÓN DE MEMORIAS PRINCIPALES DE SEMICONDUCTORES

- Activas: RAM (Estáticas y Dinámicas)
- Pasivas: ROM, PROM, EPROM, EEROM
- Otros Tipos: Burbujas Magnéticas, Sistemas Ópticos.

MEMORIAS ACTIVAS

RAM (Random Access Memory - Memoria de acceso aleatorio)

Son memorias de lectura y escritura. Su contenido puede ser modificado mediante un programa, en cualquier instante. Su mayor desventaja es la volatilidad, dado que al interrumpirse la alimentación, pierde la información almacenada en su interior.

Se utilizan como el lugar donde se cargan los programas y los datos del usuario para ser procesados.

Memorias RAM estáticas o SRAM

Está construida mediante circuitos Flip-Flop (o Biestables), en un circuito integrado.

Las memorias estáticas se llaman así, por su propiedad de mantener y conservar la información durante todo el tiempo en que se mantengan las condiciones estables de su funcionamiento.

Memorias RAM dinámicas o DRAM:

Su construcción se basa en la característica de un capacitor de retener durante un breve período de tiempo una carga eléctrica, por lo que es utilizado como un punto de memoria. Debido a su tamaño pequeño y su geometría se logran grandes densidades de integración (varios Mbits) y buenas velocidades. Estas memorias se denominan dinámicas debido a que su contenido varía o se pierde con el tiempo. Esta es su principal desventaja.

Cada bit se "escribe" en un pequeño capacitor, pero debido a las fugas que tiene, sus cargas se van perdiendo con el tiempo, tal es así, que en pocos milisegundos se descargan totalmente. Para evitar estas pérdidas de información almacenada, se procede a "refrescar" la memoria cada 1 o 2 milisegundos, mediante un procedimiento conocido como ***"ciclo de refresco" (Refresh Cycle***). Este consiste en leer la información almacenada en la memoria y volver a escribirla en la misma posición que ocupaba. Este proceso produce algunos inconvenientes, dado que debe efectuarse mediante un circuito especial que puede estar comandado por la unidad de control de la CPU o incluirse un circuito dentro del chip de la memoria. Estas soluciones presentan grandes complejidades en el Hardware. Otro inconveniente es que el ciclo de refresco puede interferir el tiempo de acceso o de lectura/escritura de la memoria por la CPU. De todas formas, el tiempo de incidencia de este proceso sobre el tiempo operativo, es del 1% al 5% y generalmente se busca optimizar las técnicas de refresco mediante operaciones de leer una columna (o fila) completa por vez, (dependiendo de la organización interna de la memoria) para luego regrabar la misma información en el mismo lugar.

Memoria RAM dinámica entrelazada

La memoria RAM entrelazada consiste en organizar las celdas de memoria en bancos. Cuando existen dos bancos, hay fabricantes que organizan las direcciones pares en un banco y las impares en el otro. De esta manera, mientras se lee una celda par, se refresca una impar, y viceversa, logrando mayor velocidad de acceso.

Protección De Paridad

Generalmente cada posición de memoria principal tiene asociado 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 a modo de verificación.

MEMORIAS PASIVAS
ROM (Read Only Memory)

Las Memorias Pasivas son memorias pregrabadas por el fabricante (**firmware**). Los datos contenidos en esta memoria pueden ser utilizados por el usuario, pero no pueden se modificados o borrados, por este motivo se dice que es una memoria **no volátil**.

Esta definición es válida para las ROM, PROM, EPROM (memorias pasivas reales) y para las EAROM y las EEROM (memorias que, a pesar de ser consideradas pasivas, pueden modificar su contenido mediante programas).

La ROM es una memoria inalterable, en la cual se han grabado de antemano los datos. Una vez que los datos se hayan escrito sobre un chip ROM, no pueden ser quitados ni modificados, sólo pueden ser leídos.

A diferencia de la memoria RAM, la ROM conserva su contenido incluso cuando el ordenador se apaga.

En la ROM se encuentran almacenados pequeños programas vitales para el funcionamiento del ordenador (controladores básicos de entrada/salida, rutinas de testeo del hardware, rutinas de carga del sistema operativo básico, juego de caracteres básicos, etc.).

Además, las ROM se utilizan extensivamente en calculadoras y dispositivos periféricos tales como impresoras láser.

Debido a la imposibilidad de modificar el contenido de las ROM, se han desarrollado alternativas denominadas genéricamente **Flash-ROM.** En este tipo de memorias el usuario puede, mediante un hardware y un software especial, programar una o varias veces el contenido de acuerdo a sus necesidades. Pertenecen a esta categoría las memorias PROM, EPROM, RPROM, EAROM y EEPROM.

PROM (Programmable ROM)

Una variante de la ROM es la memoria PROM.

Estas memorias se fabrican con chips en blanco, que se programan previamente mediante una máscara elaborada por el fabricante a pedido del usuario. Este tipo memoria no es volátil, puesto que se han depositado previamente y en forma permanente, los “unos” y los “ceros” (lógicamente hablando) dentro de cada celda de la memoria. Las ROM de máscara están destinadas a grandes volúmenes de producción.

MEMORIAS CACHÉ

MEMORIA CACHÉ INTERNA

Es una caché que se encuentra dentro del procesador, y que por su proximidad al lugar donde se cumplen todos los procesos, es mucho más rápida que la externa.

MEMORIA CACHÉ EXTERNA

Es una memoria estática que se agrega para mejorar el rendimiento de los equipos.

En una caché externa se almacenan los últimos datos e instrucciones utilizados y los datos que estadísticamente tengan mayor posibilidad de ser usados próximamente.

Mientras más se acceda, mayor será la velocidad de procesamiento, y mientras mayor sea su capacidad, habrá más posibilidad de encontrar lo que se busca.

MEMORIA SECUNDARIA

LA MEMORIA AUXILIAR

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 principal 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).

ADMINISTRADOR DE MEMORIA

INTRODUCCIÓN

Para que un proceso se pueda ejecutar necesita **memoria**. Un **módulo muy importante** del SO es el que **Administrador de Memoria** o **Gestor de Memoria**.

El **Administrador de Memoria** es un módulo perteneciente al **Administrador de Recursos** que conforma la segunda capa del SO.

FUNCIONES DEL ADMNISTRADOR DE MEMORIA

La gestión de memoria mediante un gestor favorece el rendimiento del sistema. El principal objetivo es asignar una cantidad de memoria física a los distintos procesos que la soliciten.

LA MEMORIA ES UN RECURSO VITAL.

Dentro de la gestión de memoria vamos a ver 3 aspectos fundamentales:

1) ORGANIZACIÓN

Hay que saber si el sistema va a ejecutar un proceso de usuario o varios procesos de usuario. Si la memoria puede contener varios procesos hay que saber cómo se va a dividir.

2) MANEJO DE MEMORIA

También es necesario saber cómo se lleva la cuenta del espacio de memoria ocupado y del espacio que queda libre.

Existen distintas estrategias en el manejo de memoria:

1. ***Estrategia de ubicación**:* o **asignación de memoria**, cuando un proceso llega al sistema donde se va a ubicar.
2. ***Estrategia de sustitución**:* en caso de falta de memoria, que datos y programas vamos a sacar de memoria para meter otros.
3. ***Estrategia de captura**:* como recuperar parte de un programa y sus datos en caso de que los requiera para su ejecución. Podemos distinguir 2 casos:
   - 3.1. ***Por demanda**:* sólo cuando el proceso realmente necesite cargar esa parte del espacio de direcciones para su ejecución, se cargará.
   - 3.2. ***Por anticipación**:* el sistema previamente va a cargar las partes del proceso antes de que las necesite para su ejecución.

3) PROTECCIÓN

Tenemos que proteger la zona de memoria reservada al SO de los accesos indebidos por parte de los procesos de usuario así como los espacios de direcciones de los procesos del resto de los procesos de usuario. El SO también se encarga de la compartimentación del espacio de los distintos procesos. El rendimiento del sistema se ve afectado por la eficacia del módulo del gestor de memoria.

Vamos a medir la eficacia centrándonos en dos aspectos fundamentales:

1. ***Memoria desaprovechada**:* o fragmentación de la memoria principal no utilizada, que puede provocar que el sistema sea incapaz de asignar memoria a los procesos que la solicitan.

2. ***Tiempo dedicado por el sistema a la asignación de memoria**:* si se usan algoritmos demasiado complejos, el módulo del gestor de memoria hace que el rendimiento del sistema sea más pobre.

Un programa escrito en un lenguaje fuente no es un programa ejecutable hasta que no se traduce en código máquina y no será un proceso hasta que no se le asigne memoria.

Tenemos un programa con una zona de código y otra de datos. Cuando cargamos el programa en memoria creamos un PCB, el código, datos y la pila. A éste conjunto lo llamamos **imagen del proceso.** Un proceso no será ejecutable hasta que no tenga una imagen.

En un sistema multiprogramado, el usuario no conoce donde se va a cargar el programa, depende del espacio libre de memoria principal. Se crean programas **reubicables**.

Concepto de reubicación

Es la capacidad de cargar un programa en un lugar arbitrario de la memoria en oposición de un conjunto de direcciones fijada en tiempo de traducción.

¿Cómo se carga un programa en memoria principal?. Una forma utilizada es la **carga absoluta**. Creamos un espacio de direcciones del programa que será físico, es decir, direcciones reales en memoria. Esto no es eficiente porque no podemos saber que posibles direcciones de memoria están libres.

Lo que hacemos es que cada vez que se compila el programa, se crea un espacio de direcciones lógico. Cada vez que carguemos el programa ejecutable tiene que acceder a posiciones físicas de memoria. Los cargadores son relativos ya que trabajan con espacio de direcciones lógico.

Dependiendo de cómo se realice la traducción de direcciones lógicas a físicas, tendremos 2 tipos de reubicación:

- ***Estática**:* se hace la traducción en el momento de la carga, cuando el gestor de memoria encuentra un sitio se carga el programa.

Dirección física = dirección lógica + dirección de comienzo

- ***Dinámica*:** se realiza la traducción de direcciones en tiempo de ejecución. El programa se carga en memoria sin traducir sus direcciones. Las traducciones de direcciones lógicas a físicas se van realizando a medida que se ejecuta el programa. Este tipo de traducción necesita soporte de **hardware** denominado **Memory Manager Unit (MMU).**

Dirección física = dirección lógica + contenido del registro base

En resumen:

- Tiempo de compilación à cargador absoluto.
- Tiempo de carga à reubicación estática.
- Tiempo de ejecución à reubicación dinámica.

Unidad De Manejo De Memoria (MMU)

La **MMU** un **chip** cuya labor consiste en llevar un registro de las partes de memoria que se estén utilizando y aquellas que no, con el fin de asignar espacio en memoria a los procesos cuando estos la necesiten y liberándola cuando terminen, así como administrar el intercambio entre la memoria principal y el disco en los casos en los que la memoria principal no le pueda dar capacidad a todos los procesos que tienen necesidad de ella.

Sus funciones son:

- Convertir las direcciones lógicas emitidas por los procesos en direcciones físicas.

- Comprobar que la conversión se puede realizar. La dirección lógica podría no tener una dirección física asociada. Por ejemplo, la página correspondiente a una dirección se puede haber trasladado a una zona de almacenamiento secundario temporalmente.
- Comprobar que el proceso que intenta acceder a una cierta dirección de memoria tiene permisos para ello.
- La MMU se inicializa para cada proceso del sistema. Esto permite que cada proceso pueda usar el rango completo de direcciones lógicas (memoria virtual), ya que las conversiones de estas direcciones serán distintas para cada proceso.
- En todos los procesos se configura la MMU para que la zona del núcleo (kernel) solo se pueda acceder en modo privilegiado del procesador.
- La configuración correspondiente al espacio de memoria del núcleo es idéntica en todos los procesos.

GESTIÓN DE MEMORIA

INTRODUCCIÓN

La memoria es uno de los principales recursos de la computadora, la cual debe de administrarse con mucho cuidado. Aunque actualmente la mayoría de los sistemas de cómputo cuentan con una alta capacidad de memoria, de igual manera las aplicaciones actuales tienen también altos requerimientos de memoria, lo que sigue generando escasez de memoria en los sistemas multitarea y/o multiusuario.

Los sistemas de administración de memoria se pueden clasificar en dos tipos:

- Los que desplazan los procesos de la memoria principal al disco y viceversa durante la ejecución
- Los que no.

El propósito principal de una computadora es el de ejecutar programas, estos programas, junto con los datos a los que accede deben de estar en la memoria principal (al menos parcialmente) durante la ejecución.

Para optimizar el uso del CPU y de la memoria, el SO debe de tener varios procesos a la vez en la memoria principal, para lo cual dispone de varias opciones de administración tanto del procesador como de la memoria. La selección de uno de ellos depende principalmente del diseño del hardware.

ASPECTOS GENERALES

MEMORIA REAL

La **memoria real** o **principal** es en donde son ejecutados los programas y procesos de una computadora y es el espacio real que existe en memoria para que se ejecuten los procesos. Por lo general esta memoria es de mayor costo que la memoria secundaria, pero el acceso a la información contenida en ella es de más rápido. Solo la memoria caché es más rápida que la principal, pero su costo es a su vez mayor.

MEMORIA VIRTUAL

El término **memoria virtual** se asocia a dos conceptos que normalmente a parecen unidos:

1. El uso de almacenamiento secundario para ofrecer al conjunto de las aplicaciones la ilusión de tener más memoria RAM de la que realmente hay en el sistema.
2. Ofrecer a las aplicaciones la ilusión de que están solas en el sistema, y que por lo tanto, pueden usar el espacio de direcciones completo. Esta técnica facilita enormemente la generación de código, puesto que el compilador no tiene porque preocuparse sobre dónde residirá la aplicación cuando se ejecute.

ESPACIO DE DIRECCIONES

Los **espacios de direcciones** involucrados en el manejo de la memoria son de tres tipos:

- **Direcciones físicas:** son aquellas que referencian alguna posición en la memoria física.
- **Direcciones lógicas:** son las direcciones utilizadas por los procesos. Sufren una serie de transformaciones, realizadas por el procesador (la MMU), antes de convertirse en direcciones físicas.
- **Direcciones lineales:** direcciones lineales se obtienen a partir de direcciones lógicas tras haber aplicado una transformación dependiente de la arquitectura.

Los programas de usuario siempre tratan con direcciones virtuales; nunca ven las direcciones físicas reales.

ASIGNACIÓN DE MEMORIA

ASIGNACIÓN SIMPLE CONTIGUA

Este tipo de asignación se implementa en entornos monousuarios, y permite la ejecución de un solo programa en memoria principal.

Los primero SO carecían de gestor de memoria y era el propio usuario el que proporcionaba los controladores y administraba la memoria. Como sólo podía existir un usuario a la vez, no había problemas de protección.

Cuando se crearon los SOs la organización de la memoria se basó en dividir la memoria principal en **2 partes, una** para que residiera el **SO** y **otra** para **procesos de usuarios** (procesos transitorios).

Se pensó poner el SO en ROM pero el problema es que entonces no sería modificable.

Otra de las opciones manejadas fue la de poner el controlador de dispositivos en ROM, el SO en RAM (zona baja) y la zona alta para procedimientos de usuario, tal como muestra la figura:

Protección de memoria

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

Existen dos alternativas:

1. Utilizar un límite dado por un **registro fijo por hardware**, o un **registro de protección cargado por el SO** y que sirve como punto de comienzo del proceso. Presenta la desventaja de necesitar recompilar el programa si se modifica el registro.
2. La otra opción consiste en manejar un **Registro Límite** o **Registro Base** (o de **Reubicación**)

En ambas alternativas, se almacena en el registro la posición de memoria donde se encuentra la frontera del SO y la zona de usuario.

Cuando se genera una dirección, la CPU compara la dirección a la que quiere acceder con el valor del registro límite. Si la dirección generada es mayor o igual que el registro límite, entonces es correcta. En caso contrario, se produce un error al intentar acceder a una posición de memoria del SO.

Esta comprobación se hace mediante hardware ya que sería muy lento el hacerla mediante software.

Para este esquema de protección, el procesador tiene que funcionar en modo supervisor y usuario para poder generar interrupciones.

Es sencillo de utilizar, necesitamos un comparador, pero el rendimiento del sistema no se puede optimizar si sólo podemos tener un proceso en memoria principal

Desventajas

- **Desperdicio de recursos** tanto de la memoria no utilizada, como también el desperdicio respecto de los periféricos no usados durante la ejecución del programa.
- **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.

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:

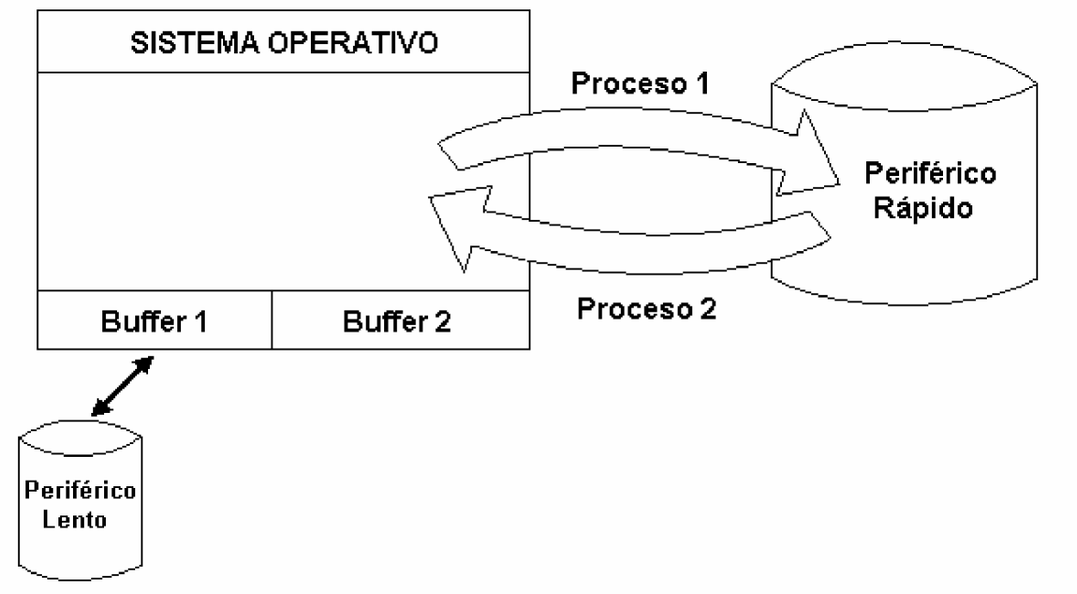
1) Simulación de multiprogramación ó Swapping

Esta técnica consiste en aprovechar los tiempos en los cuales un proceso se encuentra ocioso (Ej.: realizando una operación de E/S sobre un periférico lento) para desalojarlo completamente de la memoria, y trasladarlo al sistema de almacenamiento secundario (sobre un periférico rápido), y cargar desde el almacenamiento secundario otro proceso que estuviese pronto para su ejecución.

Obviamente esta técnica implica la necesidad de preservar buffers de memoria para recibir/emitir datos de entrada/salida para cada proceso que conviva en un determinado momento dentro del sistema.

Son necesarios además, elementos que controlen las operaciones de E/S, para permitir que el procesador pueda dedicarse a la ejecución de programas.

Actualmente esta técnica es utilizada también en sistemas operativos que manejan multiprogramación cuando, por algoritmo, se decide castigar a un proceso por haber excedido la utilización de algún recurso (en particular el procesador).

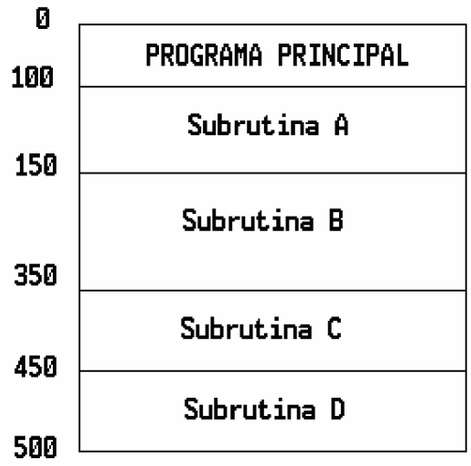


2) Simulación de mayor direccionamiento a memoria ó Técnica de Overlay

Esta técnica, también llamada **Técnica de Recubrimientos** o **Solapes** aparece porque los datos de un proceso también están cargados en memoria principal para que el proceso se pueda ejecutar. El problema es que puede que no exista bastante memoria principal para albergar todos los datos de un proceso.

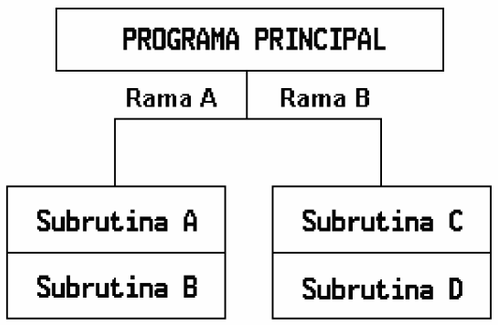
La técnica consiste en aprovechar la modularidad de los programas y en particular en la capacidad de detectar conjuntos de módulos o subrutinas que serán ejecutados en forma disjunta.

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 figura.

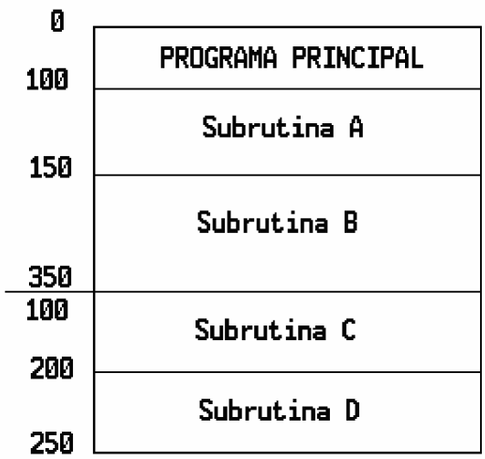


Este programa que no podrá ser ejecutado en una memoria de capacidad 400.

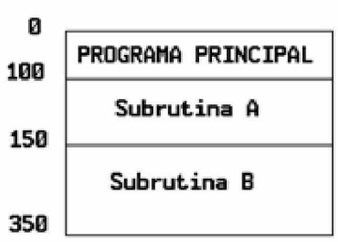
Ahora bien, si el programa pudiese ser descompuesto en forma lógica de la manera que se ve en la figura siguiente,



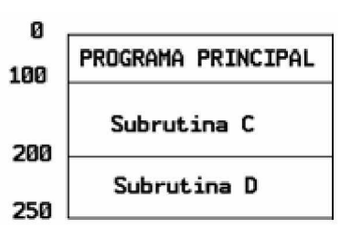
o sea que cuando se utilicen las subrutinas **Subrutina A** y **Subrutina B** (**Rama A**), no se utilizan las **Subrutina C** y **Subrutina D** (**Rama B**), podría vincularse a este programa reutilizando direcciones en la forma que se muestra en a continuación,



lo que permitiría que en algún momento se estuviese ejecutando la **Rama A** con su correspondiente mapa de memoria



y en otro la **Rama B** con un mapa de memoria como se ve a continuación



con lo cual ahora sí es posible ejecutar este programa en una memoria de capacidad 400.

En la **Técnica de Overlays** hay una porción de código y datos de usuario que deben permanecer en memoria principal durante toda la ejecución. El resto de las partes se irán metiendo en memoria principal según su necesidad

Un problema es que la ejecución puede ser muy lenta ya que tiene que hacer continuas llamadas a overlays y si éstos no están bien diseñados, el sistema se satura.

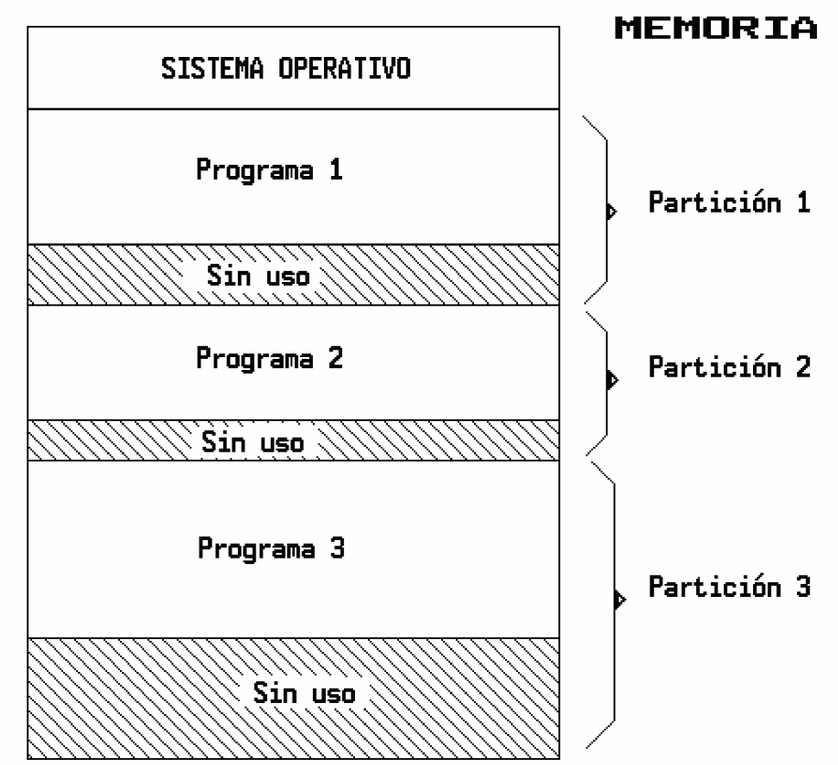
El SO se encarga de las E/S (cargar y descargar overlays en memoria principal). La ejecución del programa es más lenta que si el proceso completo se encontrara en memoria principal.

ASIGNACIÓN DE PARTICIONES FIJAS

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

En este tipo de gestión de memoria, la asignación debe ser contigua. No se pueden asignar dos particiones a un mismo proceso de usuario, y cada partición puede albergar a un solo proceso aunque sobre memoria.

Éste esquema es válido para sistemas estáticos donde se puede prever el tamaño de los procesos que se van a ejecutar.



Dentro de este esquema podemos diferenciar dos modelos:

1) Asignación de Partición Simple

En este esquema todas las particiones son iguales.

Para saber el número de particiones asignadas y las que están libres, el sistema utiliza una tabla donde se guarda el inicio de la partición, el estado (libre u ocupada) y por que proceso está ocupada.

Partición	*Dirección*	*Estado*	*Proceso*

2) Asignación de Partición Múltiple

En este caso el tamaño de las particiones es variado.

Como en el caso anterior, se vale de una tabla para llevar el estado de las particiones.

Partición	*Tamaño*	*Dirección*	*Estado*	*Proceso*

Nótese que, debido a la naturaleza variable de las particiones, se ha agregado un campo (**Tamaño)** donde registrar tamaño.

Protección

Se utilizarse un esquema de **Registro de Relocalización** (**Registro Base**) y **Registro Limite para** proteger un proceso de usuario de otro y de cambios del código y datos del sistema operativo.

El **Registro de Relocalización** contiene la **dirección física más baja** del proceso; el **Registro Limite** contiene el **rango de las direcciones lógicas** cada dirección lógica debe ser menor al Registro Limite

Fragmentación

La fragmentación es la memoria que queda desperdiciada al usar los métodos de gestión de memoria.

La fragmentación puede ser:

- **Fragmentación Externa**: existe el espacio total de memoria para satisfacer un requerimiento, pero no es contigua.
- **Fragmentación Interna**: la memoria asignada puede ser ligeramente mayor que la requerida; esta referencia es interna a la partición, pero no se utiliza.

Desventajas

Independientemente de la asignación de particiones a procesos, éste esquema de memoria tiene un problema de desaprovechamiento llamado **fragmentación Interna**

ASIGNACIÓN DE PARTICIONES VARIABLES o ASIGNACIÓN DINÁMICA

En las particiones fijas se limitaba el número de procesos en memoria, además algunas particiones se desperdiciaban a causa de que el proceso era menor que la partición.

Esto se arregla con una **partición dinámica**, de manera que las particiones no se hacen de antemano sino que conforme van llegando los procesos, el sistema va realizando las particiones.

No hay desperdicio, la partición es exactamente igual al tamaño del proceso.

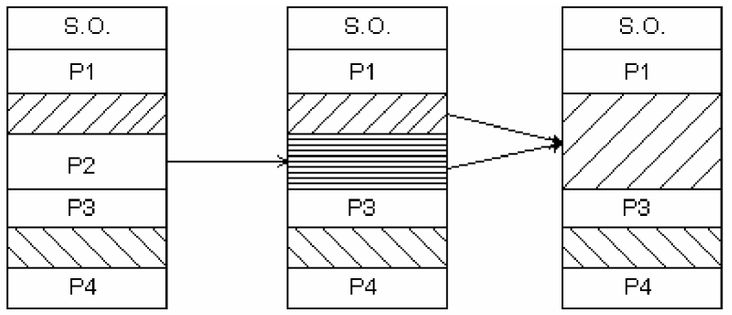
Desventajas

El problema es que cuando acaban los procesos, estos dejan huecos en el almacenamiento principal. Estos huecos pueden ser usados por otros procedimientos de igual o menor tamaño que el hueco dejado, pero aunque esto suceda, los huecos restantes van haciéndose más pequeños hasta el punto que no podrán ser asignados. Es lo que se conoce como **fragmentación externa**.

Algunas soluciones:

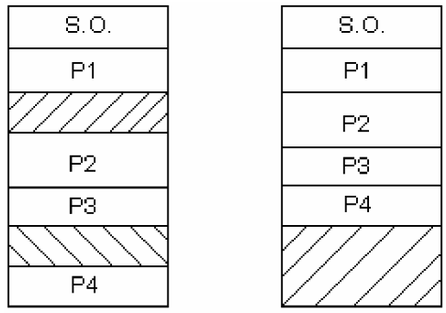
Fusión de huecos adyacentes.

Cuando termina un proceso podemos comprobar si el almacenamiento que libera tiene límites con otros agujeros y fusionarlos, formando un hueco de mayor tamaño.



Compactación de la memoria.

Consiste en trasladar las áreas ocupadas de memoria a uno de los extremos de la memoria principal, dejando a su vez un solo agujero grande de memoria libre.



Desventajas de la compactación son:

- Perdida de tiempo: durante la compactación el sistema no trabaja.
- La compactación sólo se puede realizar en el caso en que los programas sean reubicables dinámicamente.
- Si aparecen y desaparecen frecuentemente procesos, es necesaria más compactación.

Algoritmos de asignación de memoria

Se utilizan para determinar, en la memoria principal, el lugar donde serán colocados los procesos y datos que van llegando. Los 3 más utilizados son:

- ***El primer ajuste**:* al llegar un proceso nuevo, recorre la memoria desde el principio hasta encontrar el primer hueco donde quepa. Toma con rapidez la decisión del emplazamiento.
- ***El mejor ajuste**:* recorre toda la memoria hasta encontrar el hueco en el que mejor se adapte y deje el menor espacio sin usar. Es el algoritmo que mayor número de huecos pequeños deja en memoria.
- ***El peor ajuste**:* coloca el proceso en el agujero más grande posible. Así, una vez colocado sigue quedando un hueco grande para albergar un nuevo proceso.

Protección.

- ***Reubicación estática**:* cuando la CPU genera una dirección, comprueba que esa dirección no sobrepase el límite superior de la partición que está ocupando el proceso. Cuando el proceso sobrepasa la cantidad de memoria que se le ha asignado, el sistema da un error.

Registro base < Proceso < Registro límite

- ***Reubicación dinámica**:* El registro límite va a contener el número de direcciones total del proceso. El registro base va a contener la primera dirección a partir de la cual está cargando el proceso en memoria

principal. Cuando la CPU quiere acceder a memoria se compara la dirección lógica con el contenido del registro. Los registros forman parte del entorno volátil, por tanto están guardados en el PCB.

INTERCAMBIOS (SWAPPING) EN MULTIPROGRAMACIÓN

Ya hemos visto a esta técnica como una solución a la monoprogramación.

En multiprogramación se utiliza para resolver el problema de que existan en memoria principal procesos bloqueados que no pueden continuar su ejecución y que llegue un proceso nuevo y no se pueda ejecutar porque no hay espacio en memoria principal.

El **intercambio** se encarga de expropiar la memoria a procesos que se encuentran en **memoria principal** y de asignarla a procesos que se encuentran en **memoria secundaria**.

Se puede utilizar en particiones fijas, variables o sistemas monoprogramados al igual que el recubrimiento u overlay.

La técnica **swapping** no requiere que el proceso permanezca en memoria principal hasta su finalización.

Un proceso ocupa el almacenamiento principal de inmediato, se ejecuta hasta que ya no puede continuar más y cede la memoria y la CPU a otro proceso. La memoria principal se dedica a un proceso durante un breve periodo, ese trabajo es 'retirado' y entra el nuevo.

Un proceso puede ser intercambiado varias veces antes de llegar a su término.

El proceso 'retirado' de memoria principal es almacenado en un dispositivo de almacenamiento secundario rápido, donde se van a almacenar **las imágenes de los procesos** que se van a retirar temporalmente de memoria principal.

Si usamos **reubicación estática** el proceso 'retirado' deberá continuar (al entrar en memoria principal) en la misma posición de memoria de donde se sacó.

Al paso de memoria principal a disco se le llama **swapped out** y al paso de disco a memoria principal se le llama **swapped in**.

El **proceso intercambiador** o **swapper** se encarga de realizar el intercambio entre procesos.

Tiene 3 funciones:

1. **Seleccionar procesos para retirarlos de memoria principal**. Hay que salvar la imagen del proceso retirado en un dispositivo de almacenamiento secundario que sea rápido. Los procesos seleccionados para retirar de memoria principal son los procesos bloqueados y los procesos con baja prioridad.
2. **Seleccionar procesos para incorporarlos a memoria principal**. Es la función del planificador a corto plazo. Los procesos que se seleccionan para incorporación a memoria principal son aquellos que están en estado ejecutable pero residentes en memoria secundaria. Los criterios para la selección son:
   - Por la edad (cuanto tiempo ha pasado en memoria secundaria).
   - Por prioridades.
   - Si ha pasado un tiempo límite en memoria secundaria (2 seg. típicamente). Para evitar continuas operaciones de E/S el tiempo límite se basa en función del algoritmo de procesamiento.
3. **Gestionar y asignar el espacio de intercambio**.

La zona de intercambios puede ser de 2 formas:

a. ***Intercambio estático:*** se crea un espacio global en disco para todas las imágenes de los procesos que puedan ser intercambiadas en disco.

   La desventaja es que el tamaño del archivo limita el número de procesos del sistema.

   La creación de este archivo se realiza durante la inicialización por lo que no podemos tener más procesos que imágenes haya en el archivo.

   - Si el archivo es muy grande limita el tamaño en disco usado por archivos de usuario.

- Si es muy pequeño tenemos poco espacio de swapping.

b. ***Intercambio dinámico**:* cada vez que se crea un proceso se crea un espacio en el disco que va a albergar la imagen. Creamos un archivo por proceso y éste va a contener la imagen. El proceso ha de ser intercambiable.

Ventajas:

- No limita el número de procesos mientras podamos crear archivos.
- Elimina el problema del tamaño del archivo.

Desventajas:

- Requiere más espacio en disco por lo que también desperdicia más.
- Los accesos son más lentos.

Existen procesos en un SO que no deben ser intercambiados. El SO debe proporcionar mecanismos para indicar si el proceso debe ser intercambiado o no. Los mecanismos están restringidos a ciertos procesos o a ciertos usuarios. A los procesos no intercambiables no se les crea área de swapped.

MEMORIA VIRTUAL

El **espacio de direccionamiento de un sistema** (computadora) es la **capacidad de direccionamiento del hardware**, o sea la dirección a la que puede llegar con todos los bits de direccionamiento en ON, mientras que el e**spacio de direccionamiento de un proceso** es la capacidad de direccionamiento del proceso, o sea que pueda generar o pedir direcciones.

Obviamente para que un proceso ejecute en un sistema, el espacio de direccionamiento del proceso debe ser menor al espacio de direccionamiento del sistema.

Por otra parte, 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 se denomina **memoria virtual,** método que simula poseer una cantidad de memoria mayor que la existente:

Usualmente en **memoria virtual,** la memoria aparece al programador tan grande como 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.

Organización de la memoria virtual

En memoria virtual, el espacio de direcciones de memoria virtual de un proceso no ha de estar cargado en memoria principal para ejecutarse.

El hecho que en **memoria virtual** disponemos de menos memoria que la requerida para un proceso, se **soluciona** realizando **particiones del proceso de un tamaño igual al que se dispone**, cargando sólo en un instante una partición, quedando las restantes particiones en disco, de tal forma que si se desea acceder a una dirección de memoria del proceso que se encuentra en disco, se carga dicha partición de memoria.

El espacio de **direccionamiento virtual de un proceso** puede ser **mayor o igual** que el **espacio de direccionamiento real.**

La memoria virtual es útil para realizar un aumento de la multiprogramación, esto es, existencia de más procesos en el sistema, aumentando el rendimiento.

La memoria virtual requiere que se realice un **almacenamiento a dos niveles**:

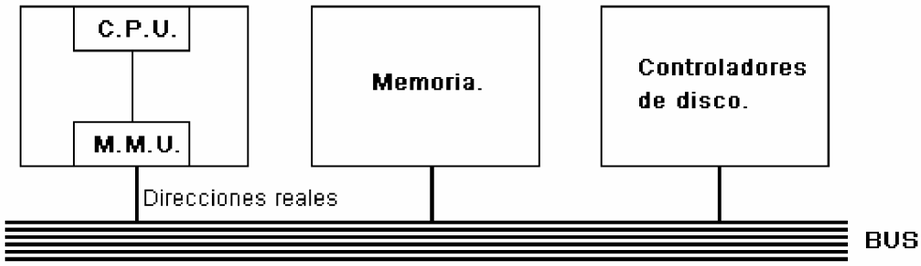
- ***Memoria real:*** donde se almacena parte del proceso.
- ***Memoria secundaria*** *(disco):* donde se almacena todo el espacio requerido por el proceso. **La imagen completa del espacio de direccionamiento virtual del proceso**.

Para realizar el almacenamiento anterior, se necesitan básicamente dos cosas:

- Se debe saber que partes de la imagen del proceso están en memoria principal y que partes no lo están.
- Debe existir una política de movimiento entre memoria principal y disco, que sea efectiva (eficiente). El hecho de tener partes del proceso en memoria secundaria va a ralentizar la velocidad de tratamiento o la ejecución del programa.

Existe un mecanismo hardware de traducciones, encargado de **traducir** de forma **simultanea** las **direcciones virtuales a reales**. Esto implica una mayor velocidad que si se realizase mediante un mecanismo software. A dicho traductor hardware se le denomina **MMU (Memory Management Unit** o **Unidad de gestión de memoria)**.

La **CPU** genera **direcciones virtuales**, por tanto, entran en el **MMU** que **genera direcciones físicas** que pasan al **bus de sistema**.



MECANISMOS DE ASIGNACIÓN DE MEMORIA VIRTUAL

Como hemos visto, cuando se utiliza memoria virtual los procesos se dividen en fragmentos, que según la técnica, pueden ser **páginas** o **segmentos**, a cada estas técnicas se las denomina **Asignación Paginada** y **Asignación Segmentada** respectivamente

ASIGNACIÓN PAGINADA

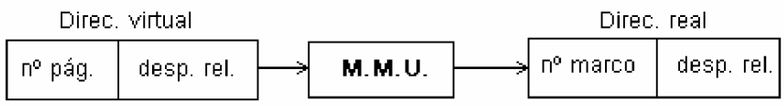
En este esquema la **memoria física** se divide en **bloques de igual tamaño**, llamados **marcos de página**, cada uno de los cuales **contendrá** una **página** de programa.

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).

La **función de paginación** es el **mecanismo** que se va a encargar de **realizar** una **correspondencia entre** los **marcos de páginas** y los **números de páginas del direccionamiento virtual** del proceso. De realizar esta función se va a encargar el **MMU**.

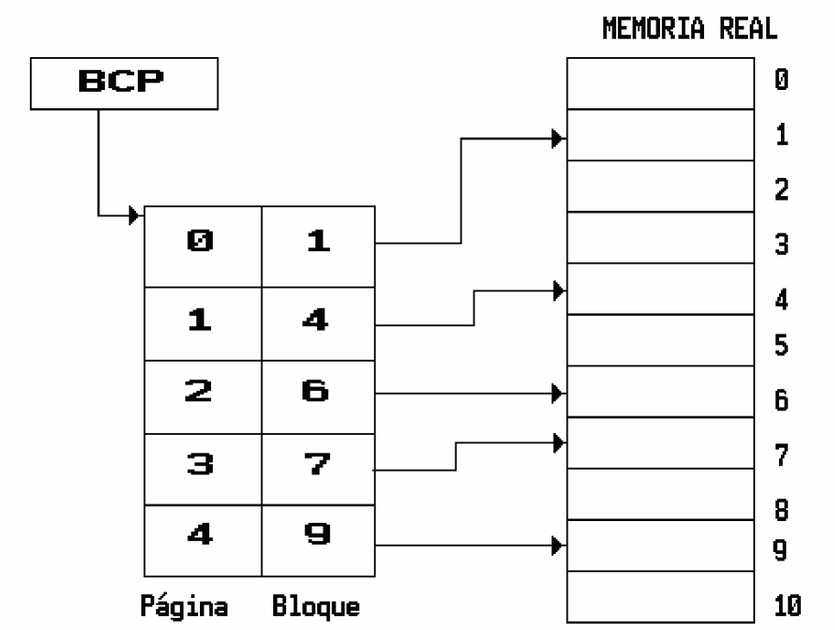
La dirección virtual se descompone en dos partes:

- **el número de página**
- **el desplazamiento dentro de la página**.



El **MMU** asocia números de página con números de marcos, siempre que la página está almacenada en memoria.

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



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:

1) la primera para llegar a la TDP y obtener la dirección del bloque correspondiente
2) y la segunda para llegar efectivamente a la dirección deseada.

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

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

La clave de protección será usada 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**.

ASIGNACIÓN PAGINADA POR DEMANDA

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

Cuando un programa es invocado se lo prepara 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)

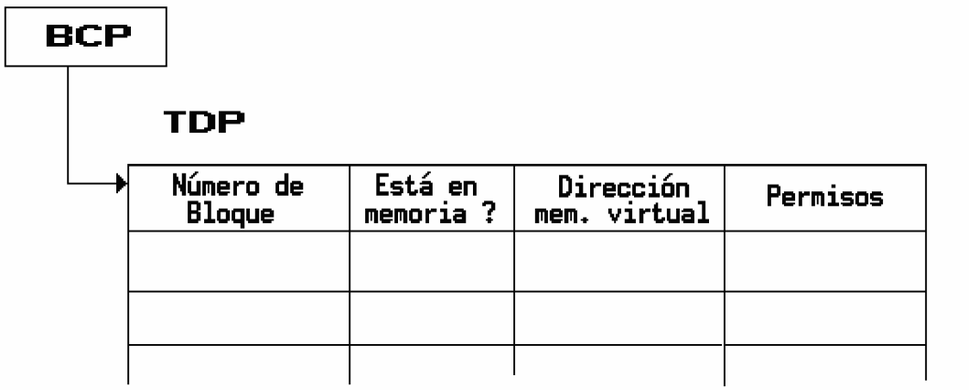


Tabla de Distribución de Bloques (una en todo el sistema)

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

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 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 estos últimos bloques suele denominárselos "en tránsito".

Desventaja de la Paginación

Fragmentación interna: al dividir el programa en páginas de tamaño fijo, si un programa ocupa menos espacio que el tamaño de una página, se estará desperdiciando el restante espacio de la página, no pudiendo ser éste asignado a otro proceso aunque cupiera (o cupiese) en dicho espacio.

ALGORITMOS DE SUSTITUCIONES LOCALES.

Estos algoritmos son los que eligen qué páginas deben ser quitadas de memoria cuando se produce una interrupción por falta de página:

ÓPTIMO.

Este algoritmo es el que menos falta de página produce. Se desplazará aquella página que no será objeto de ninguna referencia posterior. Si no hay, se desplazará aquella que se referencie más tarde.

FIFO.

Es el que más falta de página produce. En este algoritmo la página a desplazar será aquella que se cargó con mayor anterioridad.

LRU.

Es, **después del óptimo**, el que menos faltas de página produce. La página a desplazar será aquella que fue objeto de la **referencia más antigua**, esto es, sustituye aquella página de las que se encuentran en memoria que se referenció en primer lugar.

Para conseguir una buena gestión de memoria se intenta minimizar el número de faltas de página.

HIPERPAGINACIÓN

Supongamos que en un momento dado la cola de ejecutables está vacía, con esto el planificador admitiría más procesos en el sistema, conllevando esto a un aumento de la multiprogramación. Se carga el proceso en memoria principal. Supongamos que los procesos se están ejecutando y se producen faltas de página. Mientras se produce esto, los procesos van a estar bloqueados. Como el sistema detecta que la cola de ejecutables está vacía, el sistema aceptará nuevamente procesos. Así sucesivamente, tras esto se irán produciendo nuevas faltas de página de tal forma que el sistema se encontrará solamente realizando tareas de paginación.

El fenómeno que se produce cuando el sistema está continuamente realizando actividades de paginación, afectando esto al rendimiento del sistema, se denomina **hiperpaginación**.

Cuando se produce la **hiperpaginación**, los trabajos que ejecuta el sistema van a disminuir y con esto el tiempo de respuesta de cada proceso también disminuirá.

Cuando se produce **hiperpaginación**, va a aumentar la falta de página, llegando un momento en el cual el trabajo del sistema es prácticamente nulo, esto es, cae el sistema.

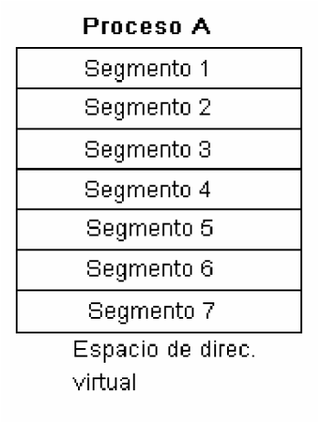
La **hiperpaginación** se puede prevenir mediante:

- **La regulación de la carga***:* es decir, mantener el grado de multiprogramación a un determinado número de procesos. Si se detecta que más del 50% del tiempo de CPU se dedica a procesos, se disminuye el grado de multiprogramación estabilizando el número de procesos en memoria.
- **Algoritmos de sustitución globales***:* miden de forma dinámica las páginas que un proceso necesita almacenar en memoria principal, para que no se produzcan faltas de página. Si necesita más páginas se asignan más marcos de página y si se necesitan menos se retiran marcos de página.

SEGMENTACIÓN

Otro de los mecanismos utilizados de asignación de memoria virtual es la **segmentación.**

Este mecanismo, al contrario que la paginación, **se basa** en la **división del espacio de direccionamiento virtual en unidades de tamaño variable** llamadas **segmentos**.



El tamaño de los segmentos es variable.

Estos segmentos **corresponden a unidades lógicas de un proceso**. Esta división es **realizada por el usuario** (**programador**).

Por ejemplo, se podría dividir un proceso en **tres segmentos**, **segmento de pila**, **segmento de datos** y **segmento de código**, donde el tamaño de cada segmento varía dependiendo de la unidad lógica que almacena. A esta división se denomina división en tiempo de ejecución.

El espacio de direcciones virtual en paginación es unidimensional, porque el usuario no tiene conocimiento de la división en páginas de dicho espacio. Por el contrario, en s**egmentación** se dice que **el espacio de direccionamiento virtual es bidimensional** porque al dividirse en segmentos, cada uno de estos comienza en la dirección cero y concluye en la última dirección del espacio, de tal forma que el programa cuando alude a un segmento, se ha de **referenciar el número de segmento** y **el desplazamiento relativo dentro del segmento**. Se ha de cumplir por esto que los elementos de un segmento se deben almacenar en **posiciones contiguas de memoria**.

Se facilita la compartición entre procesos, pero se produce el **problema de fragmentación externa**, esto es, generación de huecos que no van a poder ser realojables.

Para el alojamiento de segmentos es necesaria una asignación dinámica de memoria. Para ello se utilizan los algoritmos de particiones variables: **Mejor ajuste, primer ajuste** y **peor ajuste.**

La estructura de datos necesaria será una **tabla de segmentos** con una única entrada por cada segmento, donde se encuentran almacenados la dirección, tamaño, etc.

Se ha de almacenar la dirección de la tabla de segmentos en el descriptor de procesos.

La información que se almacenará en dicha tabla será la siguiente:

- **Dirección base del segmento:** dirección de memoria a partir de la cual comienza el segmento si está cargado en memoria principal.
- **Tamaño del segmento:** información necesaria para conocer si una dirección se encuentra fuera de las posibles direcciones de un proceso.
- **Bits de protección:** bits que aluden a la protección de escritura, lectura, etc. con el fin de permitir estas o no cuando se comparten páginas.
- **Indicador de presencia:** si el proceso está o no en memoria principal.
- **Bit de modificación*:*** indica si se ha modificado el proceso, tras su carga en memoria principal.

PASOS

1. Con el **número de segmento y el desplazamiento**, se accede a la entrada correspondiente en la **tabla de segmentos**.
2. Tras esto se mira si el proceso está en memoria principal o no.

   2.1. Si es así se comprueba si el desplazamiento es mayor que el tamaño del segmento

      2.1.1. en caso afirmativo se produce una interrupción por acceso indebido a memoria.

      2.1.2. en caso de que el desplazamiento no sea mayor que el tamaño, mediante la concatenación de la dirección de comienzo del proceso en memoria principal con el desplazamiento, obtendremos la dirección real.

   2.2. Si existe una interrupción por falta de segmento, el SO toma el control, realizando lo siguiente:

      2.2.1. Bloquea el proceso.

      2.2.2. Busca el segmento en disco, buscando la dirección de este en una tabla de ubicación en disco. La dirección de dicha tabla se almacena en el descriptor.

      2.2.3. Una vez encontrado en disco el segmento, se busca una dirección de memoria contigua, donde se pueda almacenar el segmento. En el caso de no encontrar espacio de memoria, se puede optar por realizar un desplazamiento de otro proceso o realizar una espera hasta que haya espacio para almacenar este segmento.

      2.2.4. Tras esto se carga el segmento.

      2.2.5. Por último desbloqueamos el proceso y se obtiene la dirección real.

Se suelen utilizar mecanismos para solventar problemas mixtos. Uno de estos mecanismos es la segmentación paginada.

SEGMENTACIÓN PAGINADA

Paginación y segmentación son técnicas diferentes, cada una de las cuales busca brindar las ventajas enunciadas anteriormente.

Para la segmentación se necesita que estén cargadas en memoria, áreas de tamaños variables. Si se requiere cargar un segmento en memoria; que antes estuvo en ella y fue removido a memoria secundaria; se necesita encontrar una región de la memoria lo suficientemente grande para contenerlo, lo cual no es siempre factible; en cambio "recargar" una pagina implica sólo encontrar un marco de pagina disponible.

A nivel de paginación, si quiere referenciar en forma cíclica N páginas, estas deberán ser cargadas una a una generándose varias interrupciones por fallas de páginas; bajo segmentación, esta página podría conformar un solo segmento, ocurriendo una sola interrupción, por falla de segmento. No obstante, si bajo segmentación, se desea acceder un área muy pequeña dentro de un segmento muy grande, este deberá cargarse completamente en memoria, desperdiciándose memoria; bajo paginación sólo se cargara la página que contiene los ítems referenciados.

Puede hacerse una combinación de segmentación y paginación para obtener las ventajas de ambas. En lugar de tratar un segmento como una unidad contigua, este puede dividirse en páginas. Cada segmento puede ser descrito por su propia tabla de páginas.

Los segmentos son usualmente múltiplos de páginas en tamaño, y no es necesario que todas las páginas se encuentren en memoria principal a la vez; además las páginas de un mismo segmento, aunque se encuentren contiguas en memoria virtual; no necesitan estarlo en memoria real.

Las direcciones tienen tres componentes: (s, p, d), donde la primera indica el número del segmento, la segunda el número de la página dentro del segmento y la tercera el desplazamiento dentro de la página. Se deberán usar varias tablas:

- **SMT (tabla de mapas de segmentos): una para cada proceso**. En cada entrada de la SMT se almacena la información descrita bajo segmentación pura, pero en el campo de dirección se indicara la dirección de la PMT (tabla de mapas de páginas) que describe a las diferentes páginas de cada segmento.
- **PMT (tabla de mapas de páginas): una por segmento;** cada entrada de la PMT describe una página de un segmento; en la forma que se presento la pagina pura.
- **TBM (tabla de bloques de memoria):** para controlar asignación de páginas por parte del sistema operativo.
- **JT (tabla de Jobs):** que contiene las direcciones de comienzo de cada una de las SMT de los procesos que se ejecutan en memoria.

En el caso, de que un segmento sea de tamaño inferior o igual al de una página, no se necesita tener la correspondiente PMT, actuándose en igual forma que bajo segmentación pura; puede arreglarse un bit adicional (S) a cada entrada de la SMT, que indicara si el segmento esta paginado o no.

Ventajas de la Segmentación Paginada

El esquema de segmentación paginada tiene todas las ventajas de la segmentación y la paginación:

- Debido a que los espacios de memorias son segmentados, se garantiza la facilidad de implantar la comparición y enlace.
- Como los espacios de memoria son paginados, se simplifican las estrategias de almacenamiento.
- Se elimina el problema de la fragmentación externa y la necesidad de compactación.

Desventajas de la Segmentación Paginada

- Las tres componentes de la dirección y el proceso de formación de direcciones hace que se incremente el costo de su implantación. El costo es mayor que en el caso de de segmentación pura o paginación pura.
- Se hace necesario mantener un número mayor de tablas en memoria, lo que implica un mayor costo de almacenamiento.

Sigue existiendo el problema de fragmentación interna de todas (o casi todas) las páginas finales de cada uno de los segmentos. Bajo paginación pura se desperdician solo la última página asignada, mientras que bajo segmentación – paginada el desperdicio puede ocurrir en todos los segmentos asignados.