**MEMORIA**

**William Stalling**

**7.1 REQUISITOS DE LA GESTIÓN DE LA MEMORIA**

Los requisitos que la gestión de la memoria debe satisfacer sugieren cinco requisitos:

1. Reubicación
2. Protección
3. Compartir
4. Organización lógica
5. Organización física

La administración de la memoria consiste en la división lógica de la memoria física para alojar múltiples procesos garantizando protección. Con una asignación eficiente para contener el mayor número de procesos, cuantos más procesos están en memoria más procesos competirán por la CPU. Más probabilidad que la CPU no esté ociosa.

**Reubicación**

En un sistema multiprogramado, la memoria principal disponible se comparte generalmente entre varios procesos. El programador no debe ocuparse de conocer dónde será colocado en la memoria RAM. Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones. Las referencias a la memoria se deben “traducir” según ubicación actual del proceso. De alguna forma, el hardware del procesador y el software del sistema operativo debe poder traducir las referencias de memoria encontradas en el código del programa en direcciones de memoria físicas, que reflejan la ubicación actual del programa en la memoria principal.

**Protección**

Los procesos NO deben referenciar - acceder - a direcciones de memoria de otros procesos, salvo que tengan permiso. El chequeo se debe realizar durante la ejecución, NO se puede anticipar todas las referencias que un proceso puede realizar.

Los requisitos de protección de memoria deben ser satisfechos por el procesador (hardware) en lugar del sistema operativo (software). Esto es debido a que el sistema operativo no puede anticipar todas las referencias de memoria que un programa hará.

**Compartición**

Permitir que varios procesos acceden a la misma porción de memoria. Ej Rutinas comunes, librerías, etc.

Permite un mejor uso - aprovechamiento - de la memoria RAM, evitando copias innecesarias de instrucciones

**Espacio de direcciones**

Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos.

El tamaño varía dependiendo de la arquitectura del procesador (32 o 64 bits)

Debe ser independiente de la ubicación “real” del proceso en la memoria RAM.

**Direcciones lógicas**

Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria. Representa una dirección en el “Espacio de Direcciones del Proceso”. En caso de usar direcciones lógicas, es necesario algún tipo de conversión a direcciones físicas.

**Direcciones físicas**

Referencia una localidad en la memoria principal (RAM). Dirección absoluta.

**Conversión de Direcciones**

Una forma simple de hacer esto es utilizando registros auxiliares

**Registro base**

* Dirección de comienzo del espacio de direcciones del proceso en la RAM.

**Registro límite**

* Dirección del final del proceso o medida del proceso - Tamaño de su espacio de direcciones.

Ambos valores se fijan cuando el Espacio de Direcciones del proceso es cargado a memoria. Varían entre procesos (Context Switch).

Se utiliza la dirección lógica junto con el registro base para obtener una dirección física (según la técnica utilizada). El resultado se compara con el valor del registro límite (según la técnica utilizada). Si la dirección generada es incorrecta, se genera un trap al SO.

**Binding de direcciones**

Las direcciones en los programas fuentes son simbólicas. Compiladores relacionan direcciones simbólicas con direcciones reubicables. El linkeditor/cargador relaciona las direcciones reubicables en direcciones absolutas.

**Direcciones Físicas/Lógicas y el Swapping**

Un proceso puede ser temporalmente sacado de la memoria (swapped out) a un disco de manera de permitir la ejecución de procesos.

Si se descarga considerando las direcciones físicas. Al hacer swapped in se debe cargar en el mismo espacio de memoria que ocupaba antes

Si se descarga considerando las direcciones lógicas. Al hacer swapped in se puede cargar en cualquier espacio de direcciones de memoria.

**Direcciones Lógicas vs Físicas**

Si la CPU trabaja con direcciones lógicas, para acceder a memoria principal, se deben transformar en direcciones físicas. Resolución de direcciones: transformar la dirección lógica en la dirección física correspondiente.

Resolución en momento de compilación y en tiempo de carga. Direcciones lógicas y físicas son idénticas. para reubicar un proceso es necesario recompilarlo o recargarlo.

Resolución en tiempo de ejecución. Direcciones lógicas y físicas son diferentes. Direcciones Lógicas son llamadas “Direcciones virtuales”. La reubicación se puede realizar fácilmente. El mapeo entre “virtuales” y “físicas” es realizado por hardware (Memory Management Unit - MMU)

**Memory Management Unit (MMU)**

Dispositivo de Hardware que mapea direcciones virtuales a físicas

* Es parte del procesador
* Re-programar el MMU es una operación privilegiada. Solo puede ser realizada en modo kernel

El valor en el “registro de relocación” es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria. Los procesos nunca usan direcciones físicas.

**7.2 PARTICIONAMIENTO DE LA MEMORIA** (311)

La operación principal de la gestión de la memoria es traer los procesos a la memoria principal para que el procesador los pueda ejecutar. En casi todos los sistemas multiprogramados modernos, esto implica el uso de un esquema sofisticado denominado memoria virtual. Por su parte, la memoria virtual se basa en una o ambas de las siguientes técnicas básicas: segmentación y paginación.

**Asignación de memoria**

La memoria principal debe ser organizada para contener el S.O (rutinas, librerías, estructuras) y procesos.

La memoria del S.O debe protegerse del acceso a memoria de los procesos.

La memoria de un proceso debe protegerse del acceso a memoria de otros procesos.

El esquema utilizado es dependiente del diseño del hardware

**Esquemas de asignación de memoria**

* Única partición
  + -Los procesos ocupan una única partición de memoria
  + La protección se implementa por un “límite” y un registro de “reubicación”
* Múltiples particiones
  + La memoria es dividida en varias regiones
  + Los procesos se colocan en las particiones según su tamaño
  + Técnicas:
    - Particiones fijas
    - Particiones dinámicas

**Particiones fijas**

* Regiones definidas con límites fijados
  + Particiones de IGUAL tamaño:
    - Cualquier proceso cuyo tamaño es menor igual que la partición puede ser colocado en una partición libre.
    - Si todas estan ocupadas → Swap
    - Ineficiencia: Programa más grandes que el tamaño de las particiones no podrán ejecutarse, por más que la memoria total sea más grande que el programa.
    - Ineficiencia: Programa pequeños, ocupan una partición desperdiciando mucho espacio (fragmentación interna)
  + Particiones DIFERENTE tamaño:
    - Evita el problema de las particiones de igual tamaño:
    - Procesos pequeños pueden usar particiones pequeñas
    - Se pueden prever algunas particiones grandes para procesos grandes.
    - Complejidad en el algoritmo de selección de partición para un proceso.
* Algoritmo de ubicación
  + Cada partición tiene una cola para encolar procesos adecuados para la misma:
    - Puede provocar que un proceso deba esperar lugar por un tamaño de partición dado mientras hay particiones más grandes libres
  + Utilizar una única cola para todos los procesos.

**Particiones dinámicas**

* Las particiones son de tamaño y número variables
* Los procesos son colocados exactamente en particiones de igual a su tamaño (generadas dinámicamente)
* Es necesario administrar la memoria para saber en cualquier momento, cuales porciones están siendo usadas y cuáles libres.

**Problemas con las particiones dinámicas**

* + Cada vez que entra y sale un proceso se genera huecos en la memoria, en los que eventualmente un proceso no podría entrar, pero si entraría si unimos todos los huecos (compactación).
  + Problema de fragmentación externa:
    - Espacio libre entre las particiones asignadas. Puede ser utilizado siempre y cuando el espacio requerido por un proceso, sea menor.

**Algoritmos de ubicación**

* Algoritmos de ubicación para esquemas de particiones dinámica:
  + First Fit
  + Best Fit
  + Next Fit
* **Best Fit**
  + Selecciona la partición más pequeña que contiene al proceso.
  + Mucho overhead en la búsqueda.
  + En particiones dinámicas: se generan muchos huecos pequeños de memoria libre (fragmentación externa)
* **First Fit**
  + Recorre las particiones libres en orden, buscando la primera que pueda alojar al proceso.
  + Es el algoritmo más sencillo
  + Suele ser mejor y más rápido que “Best Fit” y “Next Fit”.
* **Next Fit**
  + Mantiene las particiones libres como una lista circular.
  + Funciona como el First Fit solo que empieza desde la posición actual en la lista y no desde el principio.
  + Selecciona la primer particion que encuentra que pueda contener al proceso
  + Tiende a producir resultados ligeramente peores que “First Fit”
    - Utiliza más rápidamente particiones grandes que puedan existir al final de la lista.

**7.3 PAGINACIÓN**

Tanto las particiones de tamaño fijo como variables son ineficientes en el uso de la memoria; los primeros provocan fragmentación interna, los últimos fragmentación externa. Supóngase, sin embargo, que la memoria principal se divide en porciones de tamaño fijo relativamente pequeños, y que cada proceso también se divide en porciones pequeñas del mismo tamaño fijo. A dichas porciones del proceso, conocidas como páginas, se les asigna porciones disponibles de memorias, conocidas como marcos, o marcos de página.

En esta técnica de paginación, no hay fragmentación externa, solo fragmentación interna en la última página.

**Paginación**

* La memoria es dividida lógicamente en pequeños trozos de igual tamaño → Marcos
* El espacio de direcciones de cada proceso es dividido en trozos de igual tamaño que los marcos → páginas
* El SO mantiene una tabla de páginas por cada proceso.
  + Contiene el marco en la que está situada cada página
  + La dirección lógica consiste en un número de página y un desplazamiento dentro de la misma.

La paginación utiliza el concepto de dirección lógica. Un registro base sencillo de direcciones no basta en esta ocasión. En su lugar, el sistema operativo mantiene una tabla de de páginas por cada proceso. La tabla de páginas muestra la ubicación del marco por cada página del proceso. Dentro del programa, cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página. Con paginación, la traducción de direcciones lógicas a físicas las realiza también el hardware del procesador. Ahora el procesador debe conocer cómo acceder a la tabla de páginas del proceso actual. Presentado como una dirección lógica (número de página, desplazamiento), el procesador utiliza la tabla de páginas para producir una dirección física (número de marco, desplazamiento). Cada entrada de la tabla de página s contiene el número del marco en la memoria principal, si existe, que contiene la página correspondiente. Adicionalmente, el sistema operativo mantiene una única lista de marcos libres de todos los marcos de la memoria que se encuentra actualmente no ocupados y disponibles para las páginas.

La paginación simple es similar al particionamiento fijo. Las diferencias son que, con la paginación, las particiones son bastante pequeñas; un programa podría ocupar más de una partición; y dichas particiones no necesitan ser contiguas.

**7.4 SEGMENTACIÓN** (325)

Un programa de usuario se puede subdividir utilizando segmentación, en la cual el programa y sus datos asociados se dividen en un número de segmentos. No se requiere que todos los programas sean de la misma longitud, aunque hay una longitud máxima de segmento. Como en el caso de la paginación, una dirección lógica utilizando segmentación está compuesta por dos partes, en este caso un número de segmento y un desplazamiento.

Debido al uso de segmentos de distinto tamaño, la segmentación es similar al particionamiento dinámico. Se necesitaría que todos los segmentos de un programa se cargan en la memoria para su ejecución. La diferencia, comparada con el particionamiento dinámico, es que con la segmentación un programa podría ocupar más de una partición, y estas particiones no necesitan ser contiguas. La segmentación elimina la fragmentación interna, pero al igual que el particionamiento dinámico, sufre de fragmentación externa. Sin embargo debido a que el proceso se divide en varias piezas más pequeñas, la fragmentación externa debería ser menor.

Mientras que la paginación es invisible al programador, la segmentación es normalmente visible y se proporciona como una utilidad para organizar programas y datos. Normalmente, el programador o compilador asigna programas y datos a diferentes segmentos. Para los propósitos de la programación modular, los programas o datos se pueden dividir posteriormente en múltiples segmentos.

El inconveniente principal de este servicio es que el programador debe ser consciente de la limitación de tamaño de segmento máximo.

Otra consecuencia de utilizar segmentos de distinto tamaño es que no hay una relación simple entre direcciones lógicas y direcciones físicas. De forma análoga a la paginación, un esquema de segmentación sencillo haría uso de una tabla de segmentos por cada proceso y una lista de bloques libre de memoria principal. cada entrada de la tabla de segmentos tendría que proporcionar la dirección inicial de la memoria principal del correspondiente segmento. La entrada también debería proporcionar la longitud del segmento, para asegurar que no se utilizan direcciones no válidas. Cuando un proceso entra en el estado Ejecutando, la dirección de su tabla de segmentos se carga en un registro especial utilizado por el hardware de gestión de la memoria.

**Segmentación**

* Esquema que soporta el “punto de vista de un usuario”
* Un programa es una colección de segmentos. Un segmento es una unidad lógica como:
  + Programa Principal, Procedimientos y Funciones, variables locales y globales, stack, etc.
* Similar a particiones dinámicas
* Todos los segmentos de un programa pueden no tener el mismo tamaño (código, datos, rutinas).
* Las direcciones Lógicas consisten en 2 partes:
  + Selector de Segmento
  + Desplazamiento dentro del segmento
* Tabla de Segmentos
  + Permite mapear la dirección lógica en física.
  + Cada entrada contiene:
    - Base: Dirección física del comienzo del segmento
    - Limit: Longitud del segmento
  + Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos
  + Segment-table length register (STLR): cantidad de segmentos de un programa

Una de las tareas más importantes y complejas de un sistema operativo es la gestión de memoria. La gestión de memoria implica tratar la memoria principal como un recurso que debe asignarse y compartirse entre varios procesos activos. Para utilizar el procesador y las utilidades de E/S eficientemente, es deseable mantener tantos procesos en memoria principal como sea posible. Además, es deseable también liberar a los programadores de tener en cuenta las restricciones de tamaño en el desarrollo de programas.

**Memoria virtual** (338)

Las 2 características de la paginación y la segmentación son la clave del comienzo de la memoria virtual con:

1. Todas las referencias a la memoria dentro un proceso se realizan a *direcciones lógicas*, que se traducen dinámicamente a direcciones físicas durante la ejecución. Lo que permite a un proceso ser llevado y traído a memoria de forma que ocupe diferentes regiones de la memoria principal en distintos instantes de tiempo durante su ejecución.
2. Un proceso puede *dividirse* en varias porciones (páginas o segmentos) y estas porciones no tienen que estar localizadas en la memoria de forma contigua durante la ejecución. La combinación de la traducción de direcciones dinámicas en ejecución y el uso de una tabla de páginas o segmentos lo permite.

Supongamos que se tiene que traer un nuevo proceso de memoria. El SO comienza trayendo únicamente una o dos porciones, que incluye la porción inicial del programa y la porción inicial de datos sobre la cual acceden las primeras instrucciones. Esta parte del proceso que se encuentra realmente en la memoria principal para, cualquier instante de tiempo, se denomina conjunto residente del proceso. Usando una tabla de segmentos o páginas, el procesador siempre es capaz de determinar si puede determinar si la referencia se encuentra en memoria principal o no. Si el procesador encuentra una dirección lógica que no se encuentra en la memoria principal generará una interrupción indicando un fallo de acceso a la memoria. El sistema operativo coloca al proceso interrumpido en un estado de bloqueado y toma el control. Para que la ejecución de este proceso pueda reanudarse más adelante, el sistema operativo necesita traer a la memoria principal la porción del proceso que contiene la dirección lógica que ha causado el fallo de acceso. Con este fin, el sistema operativo realiza una petición de E/S, una lectura a disco. Después de realizar la petición de E/S. El sistema operativo puede activar otro proceso que se ejecuta mientras el disco realiza la operación de E/S. Una vez que la porción solicitada se ha traído a la memoria principal, una nueva interrupción de E/S se lanza, dando control de nuevo al sistema operativo, que coloca al proceso afectado de nuevo en el estado listo.

Existen dos implicaciones dirigidas a mejorar la utilización del sistema:

1. Pueden mantenerse un mayor número de proceso en memoria principal. Debido a que solo vamos a cargar algunas de las porciones de los procesos a ejecutar, existe espacio para más proceso. Esto nos lleva a una utilización más eficiente del procesador porque es más probable que haya al menos uno o más de los numerosos procesos que se encuentre en el estado Listo, en un instante de tiempo concreto.
2. Un proceso puede ser mayor que toda la memoria principal. se puede superar una de las restricciones fundamentales de la programación. El SO automáticamente carga porciones de un proceso en la memoria principal cuando estas se necesitan.

Debido a que un proceso ejecuta solo en la memoria principal, esta memoria se denomina memoria real. Pero el programador o el usuario perciben una memoria potencialmente mucho más grande la cual se encuentra localizada en disco. Esta última se denomina memoria virtual. La memoria virtual permite una multiprogramación muy efectiva que libera al usuario de las restricciones excesivamente fuertes de la memoria principal.

**Principio de proximidad** (343)

Por lo general prácticamente toda la memoria principal se encontrará ocupada con porciones de procesos, de forma que el procesador y el sistema operativo tengan acceso directo al mayor número posible de procesos. Así, cuando el sistema operativo traiga una porción a la memoria, debe expulsar otra. Si elimina una porción justo antes de que vaya a ser utilizada, deberá recuperar dicha porción de nuevo casi de forma inmediata. Un abuso de esto lleva a una condición denominada hiperpaginación (*thrashing)*: el sistema consume la mayor parte del tiempo enviando y trayendo porciones de swap en lugar de ejecutar instrucciones. En esencia, el sistema operativo trata de adivinar, en base a la historia reciente, qué porciones son menos probables de ser utilizadas en un futuro cercano.

Este razonamiento se basa en la creencia del principio de proximidad. Para resumir, *el principio de proximidad* indica que las referencias al programa y a los datos dentro de un proceso tienden a agruparse. Por tanto, se resume que solo unas pocas porciones del proceso se necesitarán a lo largo de un periodo de tiempo corto. También, es posible hacer suposiciones inteligentes sobre cuáles son las porciones del proceso que se necesitaran en un futuro próximo, para evitar este riesgo.

Para que la memoria virtual resulte práctica y efectiva, se necesitan dos ingredientes. Primero, debe existir un soporte de hardware para el esquema de paginación y/o segmentación. Segundo, el sistema operativo debe incluir código para gestionar el movimiento de páginas y/o segmentos entre la memoria secundaria y la memoria principal.

**Memoria virtual con paginación**

En la paginación sencilla, cada proceso dispone de su propia tabla de páginas y todas las páginas se encuentran localizadas en la memoria principal. Cada entrada en la tabla de páginas consiste en un número de marco de la correspondiente página en la memoria principal. Para la memoria virtual basada en el esquema de paginación también se necesita una tabla de páginas. De nuevo, normalmente se asocia una única tabla de páginas a cada proceso. En este caso, sin embargo, las entradas de la tabla de páginas son más complejas. Debido a que solo algunas de las páginas de proceso se encuentran en la memoria principal, se necesita que cada entrada de la tabla de páginas indique si la correspondiente página está presente (P) en memoria principal o no. Si el bit indica que la página está en memoria, la entrada también debe indicar el número de marco de dicha página.

La entrada de la tabla de páginas incluye un bit de modificado (M), que indica si los contenido de la correspondiente página han sido alterados desde que la página se cargó por última vez en la memoria principal. Si no había ningún cambio, no es necesario escribir la página cuando llegue el momento de reemplazarla por otra página en el marco de página que actualmente ocupa. Pueden existir también otros bits de control en estas entradas. Por ejemplo, si la protección y compartición se gestiona a nivel de página, se necesitarán también los bits para este propósito.

**Estructura de la tabla de páginas**

El mecanismo básico de lectura de una palabra de la memoria implica la traducción de la dirección virtual, o lógica, consiste en un número de página y un desplazamiento, a la dirección física, consiste en un número de marco y un desplazamiento, usando para ello la tabla de páginas. Debido a que la tabla de páginas es de longitud variable dependiendo del tamaño del proceso, no podemos suponer que se encuentra almacenada en los registros. En lugar de eso, debe encontrarse en la memoria principal para poder ser accedida. Cuando un proceso en particular se encuentra ejecutando, un registro contiene la dirección de comienzo de la tabla de páginas para dicho proceso. El número de página de la dirección virtual se utiliza para indexar esa tabla y buscar el correspondiente marco de página. Este, combinado con la parte de desplazamiento de la dirección virtual genera la dirección real deseada.

En la mayoría de sistemas, existe una única tabla de página por proceso. Pero cada proceso puede ocupar una gran cantidad de memoria virtual. Por consecuencia la cantidad de memoria demandada por las tablas de página únicamente puede ser inaceptablemente grande. Para resolver este problema, la mayoría de esquemas de memoria virtual almacena las tablas de página también en la memoria virtual, en lugar de en la memoria real. Esto representa que las tablas de páginas están sujetas a paginación igual que cualquier página. Cuando un proceso está en ejecución, al menos parte de su tabla de páginas debe encontrarse en memoria, incluyendo la entrada de tabla de páginas de la página actualmente en ejecución.

**Buffer de traducción anticipada**

En principio toda referencia a la memoria virtual puede causar dos accesos a memoria física: uno para buscar la entrada de la tabla de páginas apropiada y otro para buscar los datos solicitados. De esta forma, un esquema de memoria virtual básico causaría el efecto de duplicar el tiempo de acceso a la memoria. Para solventar este problema, la mayoría de esquemas de la memoria virtual utilizan una caché especial de alta velocidad para las entradas de la tabla de página, habitualmente denominada buffer de traducción anticipada (TLB). Esta cache funciona de forma similar a una memoria caché general y contiene aquellas entradas de la tabla de páginas que han sido usadas de forma más reciente. Dada una dirección virtual, el procesador primero examina la TLB, si la entrada de la tabla de páginas solicitada está presente (acierto en la TLB), entonces se recupera el número de marco y se construye la dirección real. Si la entrada de la tabla de páginas solicitada no se encuentra (fallo en la TLB), el procesador utiliza el número de página para indexar la tabla de páginas del proceso y examinar la correspondiente entrada de la tabla de páginas. Si el bit de presente está puesto a 1, entonces la página se encuentra en memoria principal, y el procesador puede recuperar el número de marco desde la entrada de la tabla de páginas para construir la dirección real. El procesador también autorizará la TLB para incluir esta nueva entrada de tabla de páginas. Finalmente, si el bit presente no está puesto a 1, entonces la página solicitada no se encuentra en la memoria principal y se produce un fallo de acceso memoria, llamado fallo de página. En este punto el SO cargará la página necesaria y actualizada de la tabla de páginas.

**Tamaño de página**

Una decisión de diseño de hardware importante es el tamaño de páginas a usar. Hay varios factores a considerar. Por un lado, está la fragmentación interna. Evidentemente, cuando mayor es el tamaño de la página, *mayor?* (en el libro dice menor) cantidad de fragmentación interna. Para optimizar el uso de la memoria principal, sería beneficioso reducir la fragmentación interna. Por otro lado, cuando menor es la página, mayor número de páginas son necesarias para cada proceso. Un mayor número de páginas por proceso significa también mayores tablas de páginas. Para programas grandes en un entorno altamente multiprogramado, esto significa que determinadas partes de las tablas de página de los procesos activos deben encontrarse en la memoria virtual, no en la memoria principal. Por tanto, puede haber un fallo de página doble para una referencia sencilla a memoria: el primero para traer la tabla de página de la parte solicitada y el segundo para traer la página del propio proceso.

El aspecto de diseño del tamaño de página se encuentra relacionado con el tamaño de memoria física y el tamaño del programa. Al mismo tiempo que la memoria principal está siendo cada vez más grande, el espacio de direcciones utilizado por las aplicaciones también crece. Está tendencia resulta más evidente en ordenadores personales y estaciones de trabajo, donde las aplicaciones tienen una complejidad creciente.

Para un tamaño determinado de una TLB, a medida que el tamaño del proceso crece y la proximidad de referencia decrece, el índice de aciertos en TLB se va reduciendo. Bajo estas circunstancias, la TLB se puede convertir en el cuello de botella del rendimiento.

Una forma de incrementar el rendimiento en la TLB es utilizar una TLB de gran tamaño, con más entradas. Sin embargo, el tamaño de TLB interactúa con otros aspectos del diseño de hardware, por ejemplo la caché de memoria principal o el número de accesos a memoria por ciclo de instrucción.

Se ha investigado la posibilidad de utilizar múltiples tamaños de página. Los tamaños de página múltiples proporcionan la flexibilidad necesaria para el uso de la TLB de forma eficiente. Por ejemplo, regiones contiguas de memoria de gran tamaño dentro del espacio direcciones del proceso, como las instrucciones del programa, se pueden proyectar sobre un reducido número de páginas de gran tamaño en lugar de un gran número de páginas de tamaño más pequeño.

**Memoria virtual con segmentación**

La segmentación permite al programador ver la memoria como si se tratase de diferentes espacios de direcciones o segmentos. Los segmentos pueden ser de tamaño diferentes, en realidad de tamaño dinámico. Una referencia a la memoria consiste en un formato de direcciones del tipo (número de segmento, desplazamiento).

Está organización tiene un gran número de ventajas para el programador sobre los espacios de direcciones no segmentados.

1. Simplifica el tratamiento de estructuras de datos que pueden crecer.
2. Permite programas que se modifican o recopilan de forma independiente, sin requerir que el conjunto completo de programas se re-enlace y se vuelvan a cargar.
3. Da soporte a la compartición entre procesos.
4. Soporta mecanismos de protección.

**Organización**

En la exposición de la segmentación sencilla, indicamos que cada proceso tiene su propia tabla de segmentos, y que cuando todos estos segmentos se han cargado en la memoria principal, la tabla de segmentos del proceso se crea y se carga también en la memoria principal. Cada entrada de la tabla de segmentos contiene la dirección de comienzo del correspondiente segmento en la memoria principal, así como la longitud del mismo. El mismo mecanismo, una tabla segmentos se necesita cuando se están tratando esquemas de memoria virtual basados en segmentación. De nuevo lo habitual es que haya una única tabla de segmentos por cada uno de los procesos. En este caso sin embargo, las entradas en la tabla de segmentos son un poco más complejas. Debido a que solo algunos de los segmentos del proceso pueden encontrarse en la memoria principal. Se necesita un bit en cada entrada de la tabla de segmentos para indicar si el correspondiente segmento se encuentra presente en la memoria principal o no. Si indica que el segmento está en memoria, la entrada también debe incluir la dirección de comienzo y la longitud del mismo.

Otro bit de control en la entrada de la tabla de segmentos es el bit de modificado, que indica si los contenidos del segmento correspondiente se han modificado desde que se cargó por última vez en la memoria principal. Si no hay ningún cambio, no es necesario escribir el segmento cuando se reemplace de la memoria principal. También puede darse otros bits de control.

El mecanismo básico para la lectura de una palabra de memoria implica la traducción de una dirección virtual, o lógica, consistente en un número de segmento y un desplazamiento, en una dirección física, usando la tabla de segmentos. Debido a que la tabla de segmentos es de tamaño variable dependiendo del tamaño del proceso, no se puede suponer que se encuentra almacenada en un registro. En su lugar, debe encontrarse en la memoria principal para poder accederse.

**Paginación y segmentación combinadas**

Paginación y segmentación, cada una tiene sus propias ventajas. La paginación es transparente al programador y elimina la fragmentación externa, y por tanto proporciona un uso eficiente de la memoria principal. La segmentación si es visible al programador, y tiene la posibilidad de manejar estructuras de datos que crecen, modularidad, y dar soporte a la compartición y a la protección. Para combinar las ventajas de ambos, algunos sistemas por medio del hardware del procesador y del soporte del sistema operativo son capaces de proporcionar ambos.

En un sistema combinado de paginación/segmentación, el espacio de direcciones del usuario se divide en un número de segmentos. Cada segmento es, por su parte, dividido en un número de páginas de tamaño fijo, que son del tamaño de los marcos de la memoria principal. Desde el punto de vista del programador, una dirección lógica sigue conteniendo un número de segmento y un desplazamiento dentro de dicho segmento. Desde el punto de vista del sistema, el desplazamiento dentro del segmento es visto como un número de página y un desplazamiento dentro de la página incluida en el segmento.

Asociada a cada proceso existe una tabla de segmentos y varias tablas de páginas, una por cada uno de los segmentos. Cuando un proceso está en ejecución, un registro mantiene la dirección de comienzo de la tabla de segmentos de dicho proceso. A partir de la dirección virtual, el procesador utiliza la parte correspondiente al número de segmento para indexar dentro de la tabla de segmentos del proceso para encontrar la tabla de páginas de dicho segmento. Después, la parte correspondiente al número de página de la dirección virtual original se utiliza para indexar la tabla de páginas y buscar el correspondiente número de marco. Este se combina con el desplazamiento correspondiente de la dirección virtual para generar la dirección real requerida.

La entrada en la tabla de segmentos contiene la longitud del segmento. También contiene el campo base, que ahora hacer referencia a la tabla de páginas. Los bits de presente y modificado no se necesitan debido a que estos aspectos se gestionan a nivel de página. Otros bits de control si pueden utilizarse, a efectos de compartición y protección. La entrada en la tabla de páginas es esencialmente la misma que para el sistema de paginación puro. El número de página se proyecta en su número de marco correspondiente si la página se encuentra presente en la memoria. El bit de modificado indica si la página necesita escribirse cuando se expulse del marco de página actual. Puede haber otros bits de control relacionados con la protección u otros aspectos de la gestión de la memoria.

**Protección y compartición**

La segmentación proporciona una vía para la implementación de las políticas de protección y compartición. Debido a que cada entrada en la tabla de segmentos incluye la longitud así como la dirección base, un programa no puede, de forma descontrolada, acceder a una posición de memoria principal más allá de los límites del segmento. Para conseguir compartición, es posible que un segmento se encuentre referenciado desde las tablas de segmentos de más de un proceso.

**Política de reemplazo**

Cuando todos los marcos de la memoria principal están ocupados y es necesario traer una nueva página para resolver un fallo de página, la política de reemplazo determina que página de las que actualmente están en memoria va a reemplazarse. Todas las políticas tienen como objetivo que la página que va a eliminarse sea aquella que tiene menos posibilidades de volver a tener una referencia en un futuro próximo. Debido al principio de proximidad de referencia, existe a menudo una alta correlación entre el histórico de referencias recientes y los patrones de referencia en un futuro próximo. Así, la mayoría de políticas tratan de predecir el comportamiento futuro en base al comportamiento pasado. En contraprestación, se debe considerar que cuanto más elaborada y sofisticada es una política de reemplazo, mayor va a ser la sobrecarga a nivel software y hardware para implementarla.

**Algoritmos básicos**

Independientemente de la estrategia de gestión del conjunto residente, existen ciertos algoritmos básicos que se utilizan para la selección de la página a reemplazar. Los algoritmos de reemplazo que se han desarrollado a lo largo de la literatura son:

* Óptimo
* Usado menos recientemente (LRU)
* FIFO
* Reloj

La política ***óptima*** de selección tomará como reemplazo la página para la cual el instante de la siguiente referencia se encuentra más lejos. Se puede ver que para esta política los resultados son el menor número de posibles fallos de página. Evidentemente, esta política es imposible de implementar, porque requiere que el sistema operativo tenga un perfecto conocimiento de los eventos futuros.

La política de reemplazo de la página ***usada menos recientemente (LRU)*** seleccionará como candidata la página de memoria que no se haya referenciado desde hace más tiempo. Debido al principio de proximidad referenciada, está página sería la que tiene menos probabilidad de volver a tener referencias en un futuro próximo. Y, de hecho, la política LRU proporciona unos resultados casi tan buenos como la política Óptima. El problema con esta alternativa es la dificultad en su implementación. Una solución sería etiquetar cada página con el instante de tiempo de su última referencia; esto podría ser en cada una de las referencias a la memoria, bien instrucciones o datos. Incluso en el caso de que el hardware diera soporte a dicho esquema, la sobrecarga sería tremenda.

La política ***FIFO*** trata los marcos de página ocupados como si se tratase de un buffer circular, y las páginas se reemplazan mediante una estrategia cíclica de un tipo round-robin. Todo lo que se necesita es un puntero que recorre de forma circular los marcos de página del proceso. Por tanto, se trata de una de las políticas de reemplazo más sencillas de implementar. El razonamiento tras este modelo, además de su simplicidad, es el reemplazo de la página que lleva en memoria más tiempo: una página traída a la memoria hace mucho tiempo puede haber dejado de utilizarse. Este razonamiento a menudo es erróneo, debido a que es habitual que en los programas haya una zona del mismo o regiones de datos que son utilizados de forma intensiva durante todo el tiempo de vida del proceso. Esas páginas son expulsadas de la memoria y traídas de nuevo de forma repetida por n algoritmo de tipo FIFO.

Mientras que la política LRU alcanza unos resultados similares a la política óptima, es difícil de implementar e impone una sobrecarga significativa. Por otro lado, la política FIFO es muy sencilla de implementar pero su rendimiento es relativamente pobre. A lo largo de los años, los diseñadores de sistemas operativos han intentado un gran número de algoritmos diferentes para aproximarse a los resultados obtenidos por LRU e intentado imponer una sobrecarga más reducida. Muchos de estos algoritmos son variantes del esquema denominado política del reloj.

En su forma más sencilla la política ***del reloj*** requiere la inclusión de un bit adicional en cada uno de los marcos de página, denominado bit de usado. Cuando una página trae por primera vez a la memoria, el bit de usado de dicho marco se pone a 1. En cualquier momento que la página vuelve a utilizarse (después de la referencia generada con el fallo de página inicial) su bit de usado se pone a 1. Para el algoritmo de reemplazo de páginas, el conjunto de todas las páginas que son candidatas para reemplazo se disponen como si se tratase de un buffer circular, al cual se asocia un puntero. Cuando se reemplaza una página, el puntero indica el siguiente marco del buffer justo después del marco que acaba de actualizarse. Cuando llega el momento de reemplazar una página, el sistema operativo recorre el buffer para encontrar un marco con su bit de usado a 0. Cada vez que encuentra un marco con el bit de usado a 1, se reinicia este bit a 0 y se continúa. Véase que está política es similar a FIFO, excepto que, en la política del reloj, el algoritmo soltara todo marco con el bit de usuario a 1.

**Hiperpaginación**

Decimos que un sistema está en trashing cuando pasa más tiempo paginando que ejecutando procesos. Como consecuencia, hay una baja importante de performance en el sistema.

El SO monitorea el uso de la CPU, si hay baja utilización entonces aumenta el grado de multiprogramación, si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos. Cuando un proceso necesita más frames, comienzan los page-faults y robo de frames a otros procesos. Por swapping de páginas y encolamiento en dispositivos, baja el uso de la CPU.

**Hiperpaginación y scheduler de CPU**

Cuando se decrementa el uso de la CPU, el scheduler long term aumenta el grado de multiprogramación. El nuevo proceso inicia nuevos page faults, y por lo tanto, más actividad de paginado. Se decrementa el uso de la CPU.

**Control de hiperpaginación**

Se puede limitar el thrashing usando algoritmos de reemplazo local. Con este algoritmo, si un proceso entra en thrashing no roba frames a otros procesos. Si bien perjudica la performance del sistema, es controlable.

**El modelo de localidad**

Cercanía de referencia o principio de cercanía. Las referencias a datos y programas dentro de un proceso tienden a agruparse. La localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento. En cortos periodos de tiempo, el proceso necesitará pocas “piezas” del proceso (por ejemplo, una página de instrucciones y otra de datos). Un programa se compone de varias localidades. Ejemplo: Cada rutina será una nueva localidad: se referencia sus direcciones (cercanas) cuando se está ejecutando. Para prevenir la hiperactividad, un proceso debe tener en memoria sus páginas más activas (menos page faults).

**El modelo de working set**

Ya que no es posible traer todas las páginas de un proceso a la memoria principal. El SO debería saber cuanta memoria principal debería reservar para un proceso en particular. Diferentes factores entran en juego:

* Cuanto menor es la cantidad de memoria reservada para un proceso, mayor es el número de procesos que pueden residir en la memoria principal a la vez. Esto aumenta la probabilidad de que el SO pueda encontrar al menos un proceso listo para ejecutar en un instante dado.
* Si el conjunto de páginas de un proceso que está en memoria es relativamente pequeño, entonces, en virtud del principio de proximidad de referencia, la posibilidad de un fallo de página es mayor.
* Más allá de un determinado tamaño, la reserva de más memoria principal para un determinado proceso no tendrá efecto apreciable sobre la tasa de fallos de página de dicho proceso, debido al principio de proximidad de referencia.

**Políticas de asignación fija y variable**

La política de asignación fija proporciona un número fijo de marcos de memoria principal disponibles para ejecución. Siempre que se produzca un fallo de página del proceso en ejecución, la página que se necesita reemplazará una de las páginas del proceso.

Una política de asignación variable permite que se reserven un número de marcos por proceso que puede variar a lo largo del tiempo de vida del mismo. A un proceso que este causando una tasa de fallos de página relativamente alta de forma continua se le otorgaran marcos de página adicionales para reducir esta tasa de fallos.

**Ámbito de reemplazo**

La estrategia del ámbito de reemplazo se puede clasificar en global y local. Ambos tipos de políticas se activan por medio de un fallo de página cuando no existen marcos de página libres. Una política de reemplazo local selecciona únicamente entre las páginas residentes del proceso que ha generado el fallo de página. Para la identificación de la página a reemplazar en una política de reemplazo global se consideran todas las páginas en la memoria principal independientemente de a qué proceso pertenece cada página en particular mientras que las políticas locales son más fáciles de analizar no existe ninguna evidencia convincente de que proporcionen un rendimiento mejor que la política global que son más atractivas debido a la simplicidad de un de su implementación con una sobrecarga mínima.

**Estrategia del conjunto de trabajo**

El conjunto de trabajo con parámetro Δ para un proceso en el tiempo virtual ,, W(t,Δ) es el conjunto de páginas del proceso a las que se ha referenciado en las últimas Δ unidades de tiempo virtual.

El tiempo virtual se define de la siguiente manera. Considérese la secuencia de referencias a memoria r(1), r(2), .. en las cuales r(i) es la página que contiene la i-ésima dirección virtual generada por dicho proceso. El tiempo se mide en referencias a memoria; así t=1.2.3.. mide el tiempo virtual interno del proceso.

Se considera que cada una de las dos variables de W. la variable Δ es la ventana de tiempo virtual a través de la cual se observa al proceso. El tamaño del conjunto trabajo será una función nunca decreciente del tamaño de ventana. Para mayor tamaño de ventana, el tamaño del conjunto de trabajo también es mayor.