Tema3Gestiondememoria.pdf

cdl_99

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Tema 3: Gestión de memoria

📌 Estudiar en Carretero comentarios a figura 4.10

1. Introducción

1.1 Concepto de archivo

Un **archivo** es una unidad de almacenamiento lógico no vólatil que agrupa un conjunto de información relacionada entre sí bajo un mismo nombre. Desde el punto de vista del usuario,e le archivo es la única forma de gestionar el almacenamiento secundario, por o que es importante en un sistema operativo definir cómo se nombran los archivos, que operaciones hay disponibles sobre los archivos, etc.

Desde el punto de vista de SO, un archivo se caracteriza por tener una serie de atributos:

Nombre	**Identificador unico**	**Tipo de archivo**	**Mapa de archivo**
Protección	**Tamaño del archivo**	**Información temporal**	**Información de control del archivo**

Nombre de archivos

Los sistemas operativos permiten asignar un nombre único a los archivos al momento de su creación, lo que garantiza su identificación y acceso durante toda su existencia.

Estructura de archivos

Los archivos pueden tener diferentes estructuras tanto para el sistema operativo como para los usuarios. El sistema operativo necesita que ciertos archivos, como ejecutables o bibliotecas dinámicas, sigan una estructura específica para poder interpretarlos. Para los usuarios, un archivo puede organizarse como texto, registros secuenciales o indexados, entre otros. Las aplicaciones como los compiladores pueden requerir archivos con módulos y cabeceras descriptivas.

Los sistemas operativos podrían ofrecer soporte para múltiples estructuras internas de archivos, lo que facilita la programación de aplicaciones. Sin embargo, esto complica tanto la interfaz como el diseño del sistema operativo. Por eso, sistemas como UNIX o Windows optan por estructuras internas e interfaces sencillas y versátiles, permitiendo a las aplicaciones definir sus propias estructuras sin intervención del sistema operativo.

1.2 Intercambio (swapping)

Se basa en usar un disco o parte de un disco (dispositivo swap) como respaldo de la memoria principal. Cuando la memoria principal esta llena, se elgige un proceso activo y se copia en swap, esto se elige mediante un cirteorio de seleccion que varia → Ej: prioridad de proceso, tiempo que lleva en ejecucion…

> 📌 Swap: disco rápido con espacio suficiente para albergar imagenes de memoria
>
> Tiempo de transferencia: factor prinicpal en el tiempo de intercambio.
>
> El intercambiador es responsable de:
>
> - Seleccionar procesos para retirarlos de MP
> - Seleccionar procesos para incorporarlos a MP
> - Gestionar y asegurar el espacio de intercambio

Localización del espacio de intercambio:

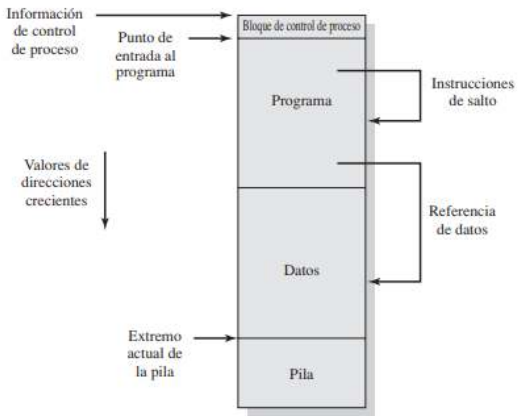
- Hay un archivo de intercambio global con la información de intercambio de todos los procesos
- Exite un archivo de intercambio para cada proceso.

2. Gestión de la memoria

2.1 Requisitos de la gestion de memoria

Reubicación

En sistemas multiprogramados, la memoria principal se comparte entre varios procesos, y los programadores no pueden predecir qué programas estarán en memoria durante la ejecución. Para optimizar el uso del procesador, es esencial permitir que los procesos se intercambien (swap) y se reubiquen en diferentes áreas de memoria.



El sistema operativo gestiona la memoria, conoce las ubicaciones clave de cada proceso (control, pila y punto de entrada) y debe traducir las direcciones de memoria usadas en el programa (como saltos e instrucciones de datos) a direcciones físicas en la memoria principal. Esto asegura que los programas puedan ejecutarse correctamente independientemente de su ubicación física.

Protección

Cada proceso debe estar protegido para evitar interferencias de otros procesos, ya sea por error o de forma intencionada. Esto implica que ningún proceso puede acceder a la memoria de otro sin permiso, tanto para lectura como para escritura.

La protección se complica porque muchos lenguajes permiten cálculos dinámicos de direcciones durante la ejecución (como índices o punteros). Por eso, todas las referencias de memoria generadas deben verificarse en tiempo real para asegurar que permanecen dentro del espacio asignado al proceso.

Afortunadamente, los mismos mecanismos que permiten la reubicación de procesos también facilitan esta protección. Además:

1. Un proceso no puede acceder al código o datos del sistema operativo.
2. No puede saltar a instrucciones ni acceder a datos de otro proceso.
3. El procesador, no el sistema operativo, se encarga de detectar y abortar accesos no permitidos, ya que prever todas las referencias de memoria sería demasiado lento y complicado.

En resumen, la protección de memoria depende del hardware (procesador) para garantizar que las referencias sean válidas durante la ejecución de las instrucciones.

Compactación

Es una técnica utilizada para **reducir la fragmentación externa** en la memoria de un sistema operativo. Como la fragmentación externa ocurre cuando hay pequeños bloques de memoria libres dispersos por todo el sistema (pero ninguno de ellos es lo suficientemente grande para satisfacer la solicitud de un nuevo proceso), la compactación trata de reorganizar esos bloques de memoria para juntar las áreas libres en un solo bloque contiguo y más grande.

Organización lógica

La memoria principal generalmente se organiza como un espacio lineal de bytes o palabras, similar a la memoria secundaria. Sin embargo, esta organización no refleja cómo se estructuran normalmente los programas, que suelen dividirse en módulos: algunos de solo lectura o ejecución y otros con datos modificables.

Si el sistema operativo y el hardware gestionan programas y datos como módulos, se obtienen varias ventajas:

1. **Desarrollo modular:** Los módulos se pueden escribir y compilar por separado, y sus referencias se resuelven en tiempo de ejecución.
2. **Protección flexible:** Se pueden aplicar diferentes niveles de protección a los módulos, como solo lectura o solo ejecución.
3. **Compartición eficiente:** Los módulos se pueden compartir entre procesos, lo que simplifica su uso porque coincide con la forma en que los usuarios conciben los problemas.

Esto hace que los sistemas sean más eficientes, seguros y fáciles de usar.

Organización física

La memoria del computador se organiza en dos niveles principales: **memoria principal** y **memoria secundaria**.

- **Memoria principal:**
  - Acceso rápido pero más costosa.
  - Volátil, lo que significa que no guarda datos de forma permanente.
  - Contiene los programas y datos en uso actualmente.
- **Memoria secundaria:**
  - Más lenta y económica.
  - No volátil, ideal para almacenamiento a largo plazo.
  - Almacena programas y datos que no se están utilizando en ese momento.

El flujo de información entre estos dos niveles es crítico, pero no se puede dejar en manos del programador porque:

1. **Memoria insuficiente:** Si la memoria principal no es suficiente, los programadores tendrían que usar técnicas como **superposición (overlaying)**, lo que complica el desarrollo y consume tiempo.
2. **Entorno multiprogramado:** En estos entornos, el programador no puede prever cuánto espacio tendrá disponible ni su ubicación.

Por eso, el sistema operativo asume la responsabilidad de mover la información entre estos niveles, lo que constituye el núcleo de la **gestión de memoria**. Esto garantiza eficiencia y simplifica el desarrollo.

2.2 Particionamiento de la memoria

La gestión de memoria se centra en cargar procesos en la memoria principal para que el procesador los ejecute. En sistemas modernos, esto se logra mediante **memoria virtual**, que utiliza técnicas como **segmentación** y **paginación**. Antes de entrar en estas técnicas avanzadas, se analizan métodos más simples, como el **particionamiento**, usado en sistemas antiguos, y las versiones básicas de paginación y segmentación, que ayudan a entender mejor el concepto de memoria virtual.

Tabla 7.1. Técnicas de gestión de memoria.

Técnica	Descripción	Virtudes	Defectos
Particionamiento fijo	La memoria principal se divide en particiones estáticas en tiempo de generación del sistema. Un proceso se puede cargar en una partición con igual o superior tamaño.	Sencilla de implementar; poca sobrecarga para el sistema operativo.	Uso ineficiente de la memoria, debido a la fragmentación interna; debe fijarse el número máximo de procesos activos.
Particionamiento dinámico	Las particiones se crean de forma dinámica, de tal forma que cada proceso se carga en una partición del mismo tamaño que el proceso.	No existe fragmentación interna; uso más eficiente de memoria principal.	Uso ineficiente del procesador, debido a la necesidad de compactación para evitar la fragmentación externa.
Paginación sencilla	La memoria principal se divide en marcos del mismo tamaño. Cada proceso se divide en páginas del mismo tamaño que los marcos. Un proceso se carga a través de la carga de todas sus páginas en marcos disponibles, no necesariamente contiguos.	No existe fragmentación externa.	Una pequeña cantidad de fragmentación interna.
Segmentación sencilla	Cada proceso se divide en segmentos. Un proceso se carga cargando todos sus segmentos en particiones dinámicas, no necesariamente contiguas.	No existe fragmentación interna; mejora la utilización de la memoria y reduce la sobrecargada respecto al particionamiento dinámico.	Fragmentación externa.
Paginación con memoria virtual	Exactamente igual que la paginación sencilla, excepto que no es necesario cargar todas las páginas de un proceso. Las páginas no residentes se traen bajo demanda de forma automática.	No existe fragmentación externa; mayor grado de multiprogramación; gran espacio de direcciones virtuales.	Sobrecarga por la gestión compleja de la memoria.
Segmentación con memoria virtual	Exactamente igual que la segmentación, excepto que no es necesario cargar todos los segmentos de un proceso. Los segmentos no residentes se traen bajo demanda de forma automática.	No existe fragmentación interna; mayor grado de multiprogramación; gran espacio de direcciones virtuales; soporte a protección y compartición.	Sobrecarga por la gestión compleja de la memoria.

Tabla resumen con las tecnicas que no usan memoria virtual

Particionamiento fijo

Es una técnica de gestión de memoria en la que la memoria principal se divide en un número fijo de particiones de tamaño predefinido. Cada partición puede alojar un proceso, y el tamaño de las particiones no cambia durante la ejecución del sistema.

Particionamiento dinamico

Es una técnica usada por los sistemas operativos para gestionar la memoria de manera flexible. A diferencia del particionamiento estático, donde la memoria se divide en bloques fijos desde el principio, en el particionamiento dinámico, la memoria se asigna de forma más flexible a medida que los programas la van necesitando.

Problemas comunes

- Fragmentacion externa: ocurre cuando hay pequeños espacios de memoria libres en diferentes partes de la memoria, pero ninguno de ellos es lo

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

suficientemente grande para satisfacer un nuevo proceso.

Solucion → Compactación de la memoria, paginación y segmentación

- Fragmenación interna: ocurre cuando un proceso solicita más memoria de la que realmente necesita, y el sistema le asigna una partición que deja espacio desperdiciado dentro de esa partición.

  Solución → mejorar asignacion de memoria, asignacion de memoria dinamica, paginacion y segmentacion

Reubicación

Es un proceso que los sistemas operativos utilizan para cambiar la ubicación en memoria de un programa o de sus componentes durante la ejecución. Esto es necesario porque, a menudo, no sabemos de antemano dónde se va a cargar un programa en la memoria, especialmente en sistemas con particionamiento dinámico o multitarea.

2.3 Paginación

La **paginación** es una técnica de **gestión de memoria** que divide la memoria en páginas de tamaño fijo para evitar la fragmentación externa y mejorar la utilización de la memoria. Al permitir que los procesos ocupen páginas dispersas por toda la memoria física, la paginación mejora la flexibilidad y eficiencia de la asignación de memoria, **aunque** puede **generar fragmentación interna** y requiere gestión adicional de las tablas de páginas.

> 📌 Paginas → porciones de procesos
>
> Marcos → porciones de memoria
>
> Tabla de paginas → muestra la ubicación del marco por cada pagina del proceso
>
> Viene expresado en 2^n

2.4 Segmentación

La **segmentación** es una técnica de gestión de memoria que divide los programas en **segmentos de tamaño variable** de acuerdo con su estructura lógica (código, datos, pila, etc.). Es más eficiente que la paginación para manejar procesos grandes y complejos y ofrece ventajas en términos de **protección** y **gestión de bibliotecas compartidas**. Sin embargo, puede generar **fragmentación externa** y requiere una gestión más compleja de la memoria.

3. Organización de la Memoria Virtual

3.1 Concepto de memoria virtual

La memoria virtual es un truco inteligente que usan los sistemas operativos para que los programas crean que tienen más memoria de la que realmente hay en el hardware (RAM).

La memoria virtual usa **almacenamiento a dos niveles**:

1. **Memoria Principal (RAM):**
   - Solo almacena las **partes activas** o necesarias de un programa en un momento específico.
2. **Memoria Secundaria (Disco):**
   - Guarda el resto del espacio del programa que no está siendo usado en ese momento.
   - Actúa como un "refuerzo" para la RAM. Cuando algo que necesitas no está en la RAM, el sistema lo trae del disco.

📌 Es necesario:

- Saber que esta en memoria principal
- Tiene una politica de movimiento entre memoria principal y memoria secundaria

La memoria virtual resuelve:

- El problema de crecimiento dinamico
- Puede aumentar el grado de multiprogramación

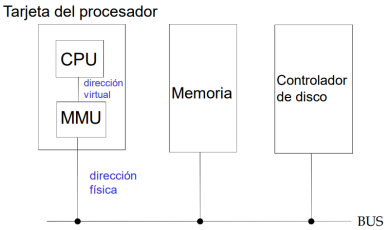
Unidad de gestión de memoria (MMU)

Es un dispositivo hardware que traduce las direcciones virtuales a direcciones fisicas. Es gestionado por el SO.

El esquema MMU es simple: el valor del registro base se añade a cada direccion generada por el proceso de usuario al mismo tiempo que es enviado a memoria. El programa usuario solo ve direcciones lógicas.

MMU tambien debe:

- Detectar si la dirección aludida se encuentra o no en MP
- Generar una exceción si no se encuentra en MP



3.2 Caracteristicas de la paginación y segmentación

1. **Direcciones lógicas y físicas**

   Todas las referencias a la memoria de un proceso se realizan mediante **direcciones lógicas**, que son traducidas dinámicamente en **direcciones físicas** durante la ejecución. Esto permite que un proceso pueda ser cargado y descargado en diferentes regiones de la memoria principal en

distintos momentos durante su ejecución, asegurando flexibilidad en la asignación de memoria.

2. **Fragmentación del proceso**

   Un proceso puede dividirse en porciones más pequeñas (como **páginas** o **segmentos**) que no necesitan estar almacenadas de forma contigua en la memoria principal durante la ejecución. Esto es posible gracias a la **traducción dinámica** y al uso de estructuras como la **tabla de páginas** o la **tabla de segmentos**.

3. **Conjunto residente**

   El **conjunto residente** se refiere a la parte del proceso que se encuentra realmente en la memoria principal en un momento dado. Este conjunto puede cambiar dinámicamente durante la ejecución del programa según las necesidades del sistema.

4. **Espacios de direcciones**

   - **Espacio de direcciones virtuales**: Está dividido en **páginas**, que son las unidades de memoria lógica. Estas páginas tienen su correspondencia en la memoria física, donde se denominan **marcos de página**.
   - **Espacio de direcciones físicas**: No tiene por qué ser contiguo. Se organiza en bloques denominados **marcos de página**.
   - **Espacio lógico**: También se organiza en bloques de tamaño uniforme llamados **páginas**.

5. **Componentes de las direcciones**

   - **Direcciones lógicas**: Generadas por la CPU, se dividen en:
     - **Número de página (p)**: Identifica qué página contiene la dirección lógica solicitada.
     - **Desplazamiento (d)**: Indica la posición específica dentro de la página.
   - **Direcciones físicas**: Se dividen en:
     - **Número de marco (m)**: Dirección base del marco en la memoria física donde está almacenada la página.
     - **Desplazamiento (d)**: Mismo que el desplazamiento en la dirección lógica.

6. **Trasiego (Thrashing)**

   El **trasiego** ocurre cuando el sistema consume la mayor parte de su tiempo trasladando porciones de memoria entre el espacio de intercambio (swap) y la memoria principal, en lugar de ejecutar instrucciones. Esto puede causar una significativa pérdida de rendimiento y se produce generalmente cuando el conjunto residente es insuficiente para las necesidades del proceso.

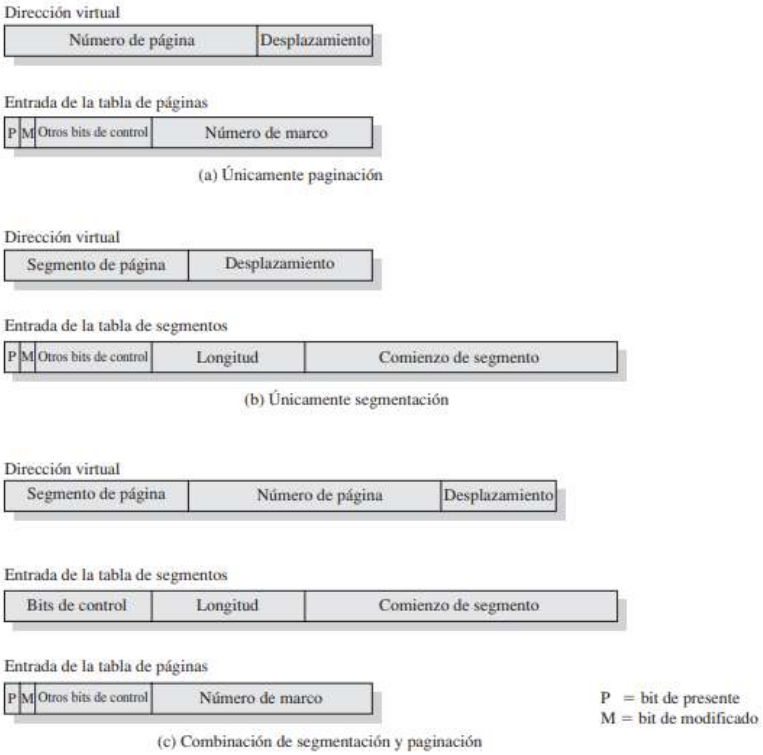


Figura 8.2. Formatos típicos de gestión de memoria.

Bit P → Indica si la página esta en memoria principal

Bit M → Indica si los contenidos de la correspondiente página han sido alterados desde que la pagina se cargo en memoria princiapl por ultima vez. Si no hay ningun cambio no es necesario escribir la pagina cuando llegue el momento de reemplazarla por otra p+agina en el marco de página que actualmente ocupa.

3.3 Paginación

En **paginación sencilla**, cada proceso tiene una tabla de páginas que indica dónde se encuentran sus páginas en la memoria principal. Todas las páginas deben estar cargadas en memoria, y cada entrada solo contiene el número de marco correspondiente.

En **memoria virtual**, no todas las páginas están en memoria al mismo tiempo, por lo que las tablas de páginas incluyen un **bit de presencia (P)**, que indica si la página está en memoria o en almacenamiento secundario. Si no está en memoria (P = 0), el sistema debe cargarla cuando se necesite, provocando una **falla de página**.

Además, las entradas pueden incluir un **bit de modificado (M)**, que señala si la página ha cambiado desde que se cargó en memoria. Si está modificada, se debe guardar en el almacenamiento secundario antes de reemplazarla. También pueden incluir bits adicionales para gestionar protección y compartición.

Cuando un proceso accede a memoria, el sistema consulta su tabla de páginas. Si la página no está en memoria, se carga desde el disco, y si no hay espacio

disponible, se reemplaza una página según criterios como el bit de modificado. Este esquema optimiza el uso de memoria física, permitiendo que los procesos trabajen con más memoria de la que realmente hay disponible.

Estructura de la tabla de páginas

El mecanismo básico para leer una palabra de memoria en sistemas con paginación consiste en convertir una **dirección virtual (o lógica)** en una **dirección física**. La dirección virtual se divide en dos partes: el **número de página** y el **desplazamiento**. Esta conversión se realiza utilizando la **tabla de páginas** del proceso.

Dado que la tabla de páginas puede tener un tamaño variable dependiendo del proceso, no es práctico almacenarla en registros. En su lugar, la tabla se encuentra en la **memoria principal**. Para acceder a ella durante la ejecución de un proceso, un registro especial guarda la dirección inicial de la tabla de páginas correspondiente.

Cuando el sistema necesita acceder a una dirección virtual:

1. Se utiliza el **número de página** como índice en la tabla de páginas.
2. La tabla devuelve el **número de marco** de memoria física donde se encuentra la página.
3. El número de marco se combina con el **desplazamiento** de la dirección virtual para formar la **dirección física** real que se utiliza para acceder a la memoria.

Por lo general, el campo del **número de página** en la dirección virtual es mayor que el campo del **número de marco** en la dirección física, ya que los procesos pueden tener un espacio de direcciones virtuales mucho más grande que la memoria física disponible.

Cada proceso tiene su propia tabla de páginas, lo que permite que ocupe una gran cantidad de memoria virtual, incluso si físicamente no está toda cargada en la memoria principal al mismo tiempo. Este diseño hace posible que los procesos trabajen con espacios de memoria mucho más grandes y complejos, gracias a la flexibilidad del mecanismo de traducción.

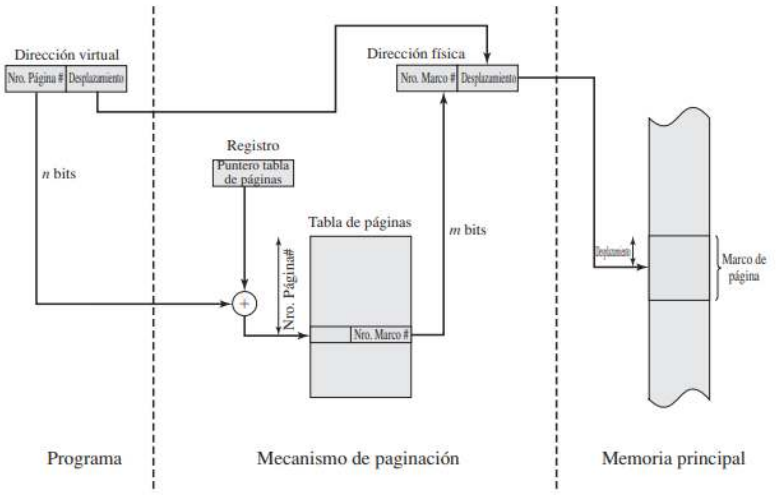


Figura 8.3. Traducción de direcciones en un sistema con paginación.

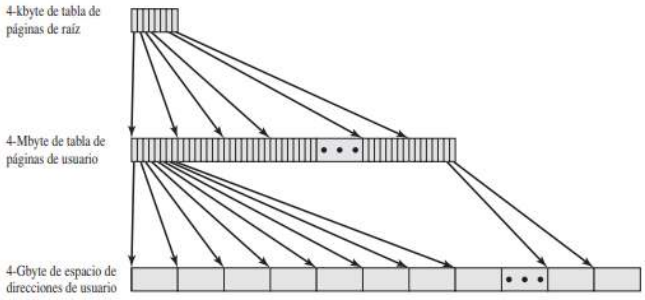


Figura 8.4. Una tabla de páginas jerárquica de dos niveles.

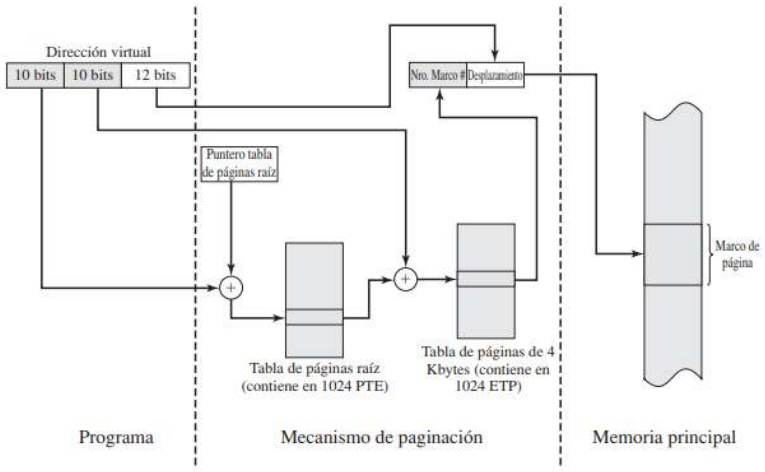


Figura 8.5. Traducción de direcciones en un sistema de paginación de dos niveles.

Buffer de traducción anticipada

El **buffer de traducción anticipada (TLB)** es una memoria caché de alta velocidad diseñada para acelerar el acceso a la memoria virtual. Sin el TLB,

cada acceso a la memoria virtual requeriría dos operaciones en la memoria física: una para consultar la tabla de páginas y otra para recuperar los datos, lo que duplicaría el tiempo de acceso.

El TLB almacena las entradas más recientes de la tabla de páginas, funcionando de manera similar a una memoria caché general. Cuando se consulta una dirección virtual:

1. El procesador verifica primero el TLB. Si la entrada está presente (**acierto en TLB**), recupera directamente el número de marco y construye la dirección física.
2. Si no está presente (**fallo en TLB**), el procesador consulta la tabla de páginas en memoria.
   - Si el bit de presencia indica que la página está en memoria, se actualiza el TLB con la nueva entrada y se genera la dirección física.
   - Si la página no está en memoria (**falla de página**), el sistema operativo interviene para cargar la página desde el almacenamiento secundario, actualiza la tabla de páginas y continúa la ejecución.

El TLB reduce significativamente el impacto de la traducción de direcciones, mejorando el rendimiento del sistema.

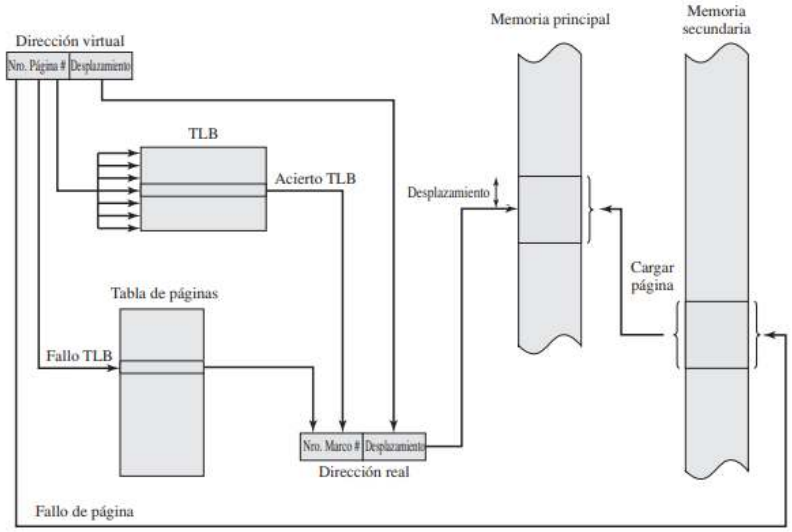


Figura 8.7. Uso de la TLB.

El mecanismo de **memoria virtual** tambien puede trabajar junto con la **caché de memoria principal** para optimizar el acceso a datos:

1. La **dirección virtual** se consulta primero en el **TLB**. Si hay acierto, se genera la dirección física combinando el número de marco con el desplazamiento.
2. Si falla, se busca en la tabla de páginas y se genera la dirección física.
3. Con la dirección física, el sistema verifica si los datos están en la caché. Si no, se recuperan de la memoria principal.

Este proceso combina TLB y caché para maximizar la eficiencia del acceso a memoria.

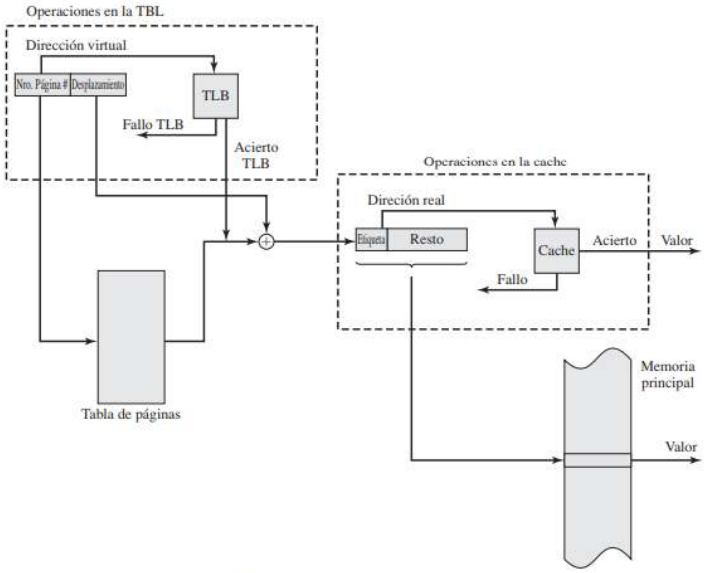


Figura 8.10. Operaciones en la TLB y en la *cache*.

¿Que hacer cuando se produce una falta de página?

- Bloquear el proceso.
- Encontrar la ubicacion en disco de la pagina solicitada.
- Encontrar un marco libre. Si no hay se opta por *swapping.*
- Cargar la pagina de disco en el marco de memoria principal.
- Actualizar las tablas.
- Desbloquear el proceso.
- Reiniciar la intrucción que origino la falta de página

Implementación de la tabla de página

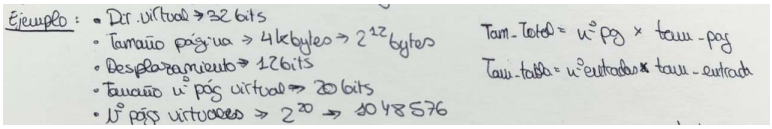
Si la tabla de paginas se encuentra en la memoria principal origina los siguientes problemas:

- Eficiencicia: para acceder a la posición deseada hay que hacer dos accesos a memoria.

  Solucion: MMU tiene un “cache” de intrucciones llamado TLB.

- Gasto de memoria: las tablas de páginas son muy grandes y hay una por cada proceso activo.

  Solución: tablas de página multinivel y tablas invertidas.



Paginación multinivel

Es una técnica que divide la tabla de páginas en varios niveles jerárquicos para gestionar eficientemente el uso de memoria en sistemas con grandes espacios de direcciones.

Existe una unica tabla del primer nivel. Cada entrada de la tabla apunta a tablas de pagina de segundo nivel y asi sucesivamente por cada nivel de la jerarquia.

Partición de tabla → permite dejar particiones no usadas sin cargar hasta que el proceso las necesita, estan no necesitan tabla de página.

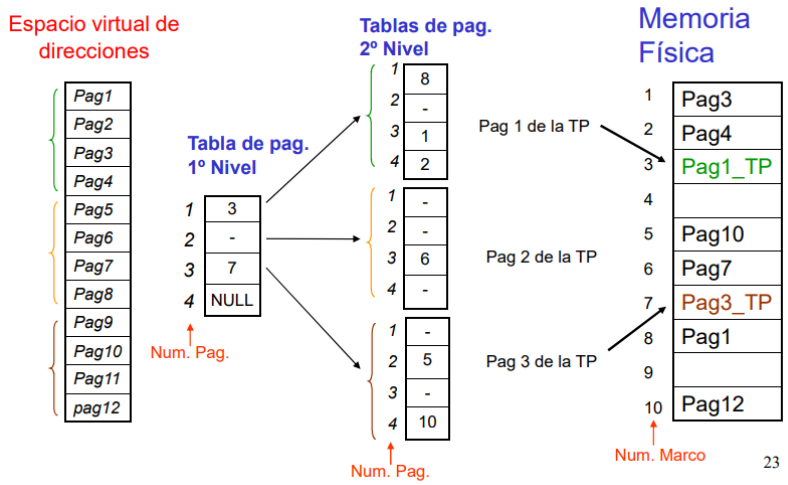
Ventajas:

- Permite compartir tablas de paginas intermedias, se ahorra espacio.
- Solo tiene que estar en memoria la tabla de paginas del primer nivel.

📌 La direccion logica es de forma: p1 p2 d

La direccion logica se divide en:

- Nº de pagina (n bits)
  - nº de pagina p1 (= k)
  - desplazamiento de pagina p2 (= n - k)
- Desplazamiento de pagina d (m bits)



3.4 Segmentación

Es una técnica de gestión de memoria que divide el espacio de direcciones de un proceso en bloques llamados **segmentos**, los cuales corresponden a diferentes

partes lógicas del programa, como código, datos, pila, etc. A diferencia de la paginación, los segmentos son de tamaños variables y reflejan la estructura lógica del programa.

Tabla de segmentos

La dirección logica esta compuesta por la dupla `<nº de segmento, desplazamiento>`

La tabla de segmentos aplcia direcciones bidimensionales definidas por el usuario en direcciones fisicas de una dimensión.

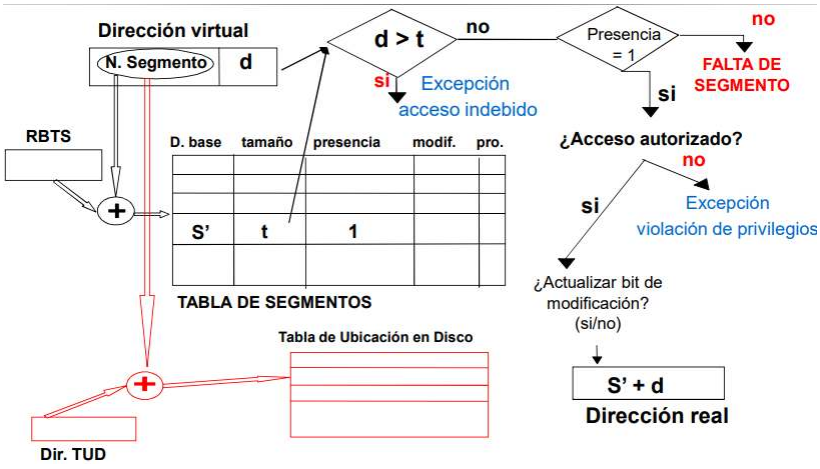
Los componentes típicos de una tabla de segmentos son:

- Base: direccion fisica donde reside el inicio del segmento de memoria
- Limite: longuitud del segmento

Cada entrada mantiene la informacion de proteccion, el registro base y los limites correspontientes a esa región. Una direccion logica con el numero de segmento y una direccion del segmento.

Este esquema presenta fragmentación externa, ua qie cada segmento se almacena en memoria de forma contigua y no todos los segmentos tienen la mism alonguitud.

Esquema de traducción



Implementación de la tabla de segmentos

- LA tabla de segmentos esta en la memoria principal.
- Registro Base de la tabla de segmentos (*RBTS*) apunta a la tabla de segmentos.
- Registro longuitus de la tabla de segmentos (*STLR*) indica el numero de segmentos del proceso.
- Nº de segmento `s`, legal si: `s < STRL`
- SO manteniene la tabla de segmentos por cada proceso y en cada cambio informa al MMU.
- SO conoce que partes de memoria estan libres.

Segmentación paginada

- Variabilidad de tamaño de segmentos y requisitos de memeoria contigua: complica la gestion de memoria
- La paginación simplifica esto pero complica compartición y protección.
- Algunos sistemas combinan ambos enfoques obteniendo las ventajas de ambos

Con la paginacion se aprovecha mejor la memoria y con la segmentacion se gestionan mejor las regiones del proceso

Con esta tecnica, cada segmento esta formado por un conjunto de paginas y no tiene que estar contiguo en memoria.

La MMU utiliza una tabla de segmentos, tal que cada entrada de la tabla apunta a una tabla de paginas.

3.5 Software del SO

El diseño de la gestión de memoria en un sistema operativo depende de tres decisiones clave:

1. Si se utiliza o no **memoria virtual**.
2. El uso de **paginación, segmentación**, o ambas.
3. Los **algoritmos** para gestionar la memoria.

La elección de paginación o segmentación depende del soporte de hardware. Por ejemplo, los primeros sistemas UNIX no usaban memoria virtual por la falta de soporte en los procesadores. Actualmente, casi todos los sistemas operativos importantes emplean memoria virtual, mientras que los sistemas de segmentación pura son raros. En la mayoría de casos, la segmentación se combina con paginación, lo que concentra la gestión de memoria en esta última técnica.

Politicas del SO para la memoria virtual

Politica de recuperación

La **política de recuperación** decide cuándo traer una página a la memoria principal. Existen dos enfoques principales:

1. **Paginación bajo demanda**: Las páginas se cargan solo cuando se accede a ellas, lo que inicialmente causa muchos fallos de página, pero se estabiliza gracias al principio de proximidad.
2. **Paginación adelantada**: Además de la página requerida, se cargan páginas contiguas de la memoria secundaria, aprovechando la eficiencia de acceso secuencial. Sin embargo, puede ser ineficiente si las páginas adicionales no se usan.

La paginación adelantada suele activarse durante fallos de página, haciéndola invisible al programador. Es diferente al **swapping**, ya que este expulsa todas las páginas de un proceso al suspenderlo y las recupera al reactivarlo.

Politica de ubicacion

La **política de ubicación** decide dónde se alojan las porciones de memoria de un proceso en la memoria principal.

- En sistemas de **segmentación pura**, esta decisión es crucial y se utilizan estrategias como mejor ajuste o primer ajuste.
- En sistemas con **paginación pura o paginación combinada con segmentación**, la ubicación es menos relevante, ya que el hardware puede gestionar cualquier combinación de página y marco con igual eficiencia.

Sin embargo, en sistemas **NUMA (memoria no uniforme)**, donde el tiempo de acceso a la memoria depende de la distancia entre el procesador y el módulo de memoria, la ubicación es crítica. Aquí, una estrategia efectiva asigna páginas al módulo de memoria que ofrezca el mejor rendimiento.

Politica de reemplazo

La **política de reemplazo** decide qué página de la memoria principal será sustituida cuando se necesita cargar una nueva página debido a un fallo de página. Este tema abarca tres aspectos clave:

1. **Gestión del conjunto residente:**
    - ¿Cuántos marcos se asignan a cada proceso?
    - ¿Se consideran solo páginas del proceso que causó el fallo o de todos los procesos?
2. **Selección de la página a reemplazar:**
    - Se elige la página menos probable de ser referenciada próximamente, basándose en el principio de proximidad de referencia, que relaciona el uso reciente con el uso futuro.

Aunque las políticas más complejas pueden predecir mejor el comportamiento, también implican mayores costos en software y hardware para su implementación.

El **bloqueo de marcos** impide que ciertas páginas almacenadas en la memoria principal sean reemplazadas. Esto aplica a áreas críticas como el núcleo del sistema operativo, buffers de E/S y otras estructuras esenciales. Para implementarlo, se asocia un bit de bloqueo a cada marco, que puede ubicarse en la tabla de marcos o en la tabla de páginas.

Los **algoritmos básicos de reemplazo de páginas** son estrategias que determinan qué página en la memoria principal será reemplazada cuando sea necesario cargar una nueva. Entre ellos se encuentran:

- **Óptimo**: Selecciona la página cuya próxima referencia está más lejos en el futuro. Aunque es el más eficiente en cuanto a reducir fallos de página, es impracticable porque requiere conocer eventos futuros con precisión.
- **LRU (Usado menos recientemente)**: Reemplaza la página que no ha sido referenciada durante más tiempo. Es una aproximación práctica del óptimo.
- **FIFO (Primero en entrar, primero en salir)**: Reemplaza la página más antigua en memoria, sin considerar su uso reciente. Es simple, pero puede ser ineficiente.
- **Reloj**: Una versión mejorada de FIFO, utiliza un puntero y un bit de uso para decidir qué página reemplazar, permitiendo un enfoque más eficiente.

El algoritmo **óptimo** se utiliza como referencia teórica para evaluar la efectividad de los demás.

Gestion del conjunto residente

El **tamaño del conjunto residente** en un sistema de memoria virtual paginada se refiere a la cantidad de memoria principal reservada para un proceso. No es necesario cargar todas las páginas de un proceso en memoria, ya que se pueden cargar solo las necesarias. Esto depende de varios factores:

- **Memoria reservada más pequeña**: Permite que más procesos residan en memoria, reduciendo el tiempo perdido por swapping y mejorando la capacidad de ejecución del sistema.
- **Memoria pequeña aumenta fallos de página**: Si el conjunto de páginas en memoria es pequeño, es más probable que ocurra un fallo de página debido al principio de proximidad de referencia.
- **Más memoria no siempre ayuda**: Después de cierto tamaño, aumentar la memoria reservada no reduce significativamente los fallos de página.

En los sistemas operativos modernos, existen dos políticas principales:

- **Asignación fija**: Se asigna un número fijo de marcos de memoria al proceso, determinado al inicio de la ejecución. Si ocurre un fallo de página, una página del proceso en memoria se reemplaza por la nueva.
- La **política de asignación variable** permite que el número de marcos de memoria asignados a un proceso cambie durante su ejecución, según su comportamiento. Si un proceso presenta una tasa alta de fallos de página, se le asignarán más marcos para reducir los fallos. Por otro lado, si tiene una tasa baja de fallos, se le reducirán los marcos, sin que esto aumente significativamente los fallos.

Aunque esta política parece más eficiente, su implementación es compleja, ya que el sistema operativo debe monitorear el comportamiento del proceso para ajustar dinámicamente la cantidad de memoria. Esto genera una sobrecarga de software y depende de los mecanismos de hardware del procesador.

El **ámbito de reemplazo** se clasifica en **global** y **local**. En una política de reemplazo local, solo se seleccionan páginas del proceso que causó el fallo de página. En cambio, una política global considera todas las páginas en memoria principal (no bloqueadas) como candidatas para el reemplazo, sin importar el proceso al que pertenezcan. Aunque las políticas locales son más fáciles de analizar, las globales pueden ser más eficientes y tienen una implementación más sencilla.

El **tamaño del conjunto residente** influye en el ámbito de reemplazo: un conjunto residente fijo implica un reemplazo local, mientras que uno variable permite un reemplazo global, ya que el tamaño del conjunto de un proceso puede cambiar. Las combinaciones son las siguientes:

1. **Asignación fija, ámbito local**: El sistema asigna un número fijo de marcos a cada proceso. Cuando ocurre un fallo de página, se reemplaza una página dentro del mismo proceso. Esta estrategia puede ser ineficiente si las reservas de memoria son muy grandes o muy pequeñas.

2. **Asignación variable, ámbito global**: Se asigna un número variable de marcos a cada proceso, y cuando ocurre un fallo de página, se puede asignar un marco libre de cualquier proceso. Este enfoque es fácil de implementar y reduce la tasa de fallos de página al permitir que los procesos crezcan según sus necesidades.

Politica de limpieza

La **política de limpieza** se encarga de decidir cuándo escribir una página modificada en memoria secundaria. Las dos alternativas principales son:

1. **Limpieza bajo demanda**: La página se escribe en memoria secundaria solo cuando se selecciona para ser reemplazada.
2. **Limpieza adelantada**: La página se escribe antes de ser reemplazada, permitiendo escribir varias páginas a la vez.

Sin embargo, ambas políticas tienen inconvenientes. En la **limpieza adelantada**, escribir páginas innecesarias puede desperdiciar recursos de memoria secundaria si las páginas se modifican nuevamente antes de ser reemplazadas. En la **limpieza bajo demanda**, el proceso que causa el fallo de página debe esperar dos transferencias (escritura y lectura), lo que reduce el rendimiento del procesador.

Una solución más eficiente es el **buffering de páginas**, que desacopla las operaciones de limpieza y reemplazo. Las páginas reemplazadas se clasifican en dos listas: modificadas y no modificadas. Las páginas modificadas se escriben en lotes, y las no modificadas pueden ser reclamadas o descartadas cuando se asignan nuevos marcos. Esto mejora la eficiencia al reducir la cantidad de transferencias innecesarias.

Control de carga

El control de carga determina el número de procesos que residirán en la memoria principal, eso se denomina el grado de multiprogramación.

El **grado de multiprogramación** se refiere a la cantidad de procesos que residen en memoria al mismo tiempo. A medida que aumenta este grado, la utilización del procesador tiende a subir, ya que hay menos posibilidades de que los procesos estén bloqueados. Sin embargo, cuando se alcanza un cierto nivel, el tamaño del conjunto residente promedio de los procesos no es adecuado, lo que provoca un aumento dramático de fallos de página y una caída en la utilización del procesador.

Cuando el grado de multiprogramación debe reducirse, algunos procesos deben suspenderse (enviarse a swap). Según **CARR84**, existen seis criterios para decidir qué procesos suspender:

1. **Procesos con baja prioridad**: No se relacionan con el rendimiento, pero tienen menor prioridad para ser suspendidos.
2. **Procesos que causan muchos fallos**: Si un proceso causa muchos fallos de página, suspenderlo mejora el rendimiento global al evitar la sobrecarga de reemplazo de páginas y operaciones de E/S.

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

3. **Procesos activados hace más tiempo**: Tienen menos probabilidades de tener su conjunto de trabajo residente, por lo que su suspensión reduce la probabilidad de un fallo adicional.
4. **Procesos con el conjunto residente más pequeño**: Estos procesos requieren menos esfuerzo para ser recargados, aunque esto penaliza a los procesos con fuerte proximidad de referencias.
5. **Procesos más grandes**: Liberan más marcos de memoria, reduciendo la necesidad de más suspensiones en el corto plazo.
6. **Proceso con la mayor ventana de ejecución restante**: Suspender procesos que han ejecutado menos tiempo puede mejorar la eficiencia, similar a la estrategia de **round robin** en planificación de procesos.

4. Gestión de la Memoria Virtual

4.1 Introducción

Gestión de memeoria virtual con paginación

Criterios de planificacion respecto a:

- Politicas de asignación (necesario determinar cuantos marcos de pagina asignan a cada proceso)
  - **Fija**: Asigna un número fijo de marcos de página a cada proceso desde su inicio.
  - **Variable**: El número de marcos puede cambiar durante la ejecución del proceso según su necesidad de memoria.
- Politicas de busqueda:
  - **Paginación por demanda**: Las páginas se cargan solo cuando se necesitan (cuando ocurre un fallo de página).
  - **Paginación anticipada**: Las páginas se cargan antes de ser necesarias, para evitar fallos de página.
- Politicas de sustitución:
  - **Sustitución local**: Solo se consideran las páginas del proceso que causó el fallo de página.
  - **Sustitución global**: Se consideran todas las páginas de la memoria para reemplazo, no solo las del proceso que causó el fallo.

📌 Criterios que siempre denebn cumplirse:

- Paginas “limpias” frente a “sucias”: minimizar el coste de transferencia
- Páginas compartidas: reducir el numero de faltas de paginas
- Paginas especiales: algunos marcos pueden estar bloquedos.

Influencia del tamaño de pagina → Busqueda de equilibrio

+ pequeña	> Aumento de tamaño de las tablas de páginas > Aumento del nºtrasnferencias entre MP y Disco > Reducen la fragmentacion interna
+ grande	• Grandes cantidades de información que no serán usadas están ocupando MP • Aumenta la fragmentación interna

4.2 Algoritmos de sustitución

Influye mas en la cantidad de MP dispo nible que el algoritmo usado.

Algorimo Optimo: debe generar el minimo nº de fallos de página. Se reemplaza la que tardaba mas tiempo en volver a usarse esto es irrealizable pues no se puede predecir. Se usa para compararlo con el resto de algoritmo

Algoritmo FIFO: Se selecciona la pagina que lleva mas tiempo en memoria. Simple y no necesita apoyo de hardware especial. Tiene que habre lista ordenada segun el tiempo. Cuando hay una nueva pagina se pone alf inal de la lista.

No siempre es bueno puede ser que la pagina que lleve mas tiempo sea importante.

Puede sufrir *Armonia de Belady:* aparentemente cuanto mas marcos de paginas haya en el sistema, menos fallos de pagina se produciran pero a veces ocurre lo contrario.

Algoritmo LRU (last recently used): basado en el principio de proximidad temporal, la pagina que se debe reemplazar es la que no se ha referenciado desde hace mas tiempo. No sufre *Anomalia de Belady.* Este tipo de algotirmos se nococe como algoritmos de pila.

Este algoritmo tiene en cuenta el tiempo lógico de cada proceso no el real. Es eficiente pero dificil de implementar, se necesita mucho apoyo hardware.

Posible **implementacion**: tener un cpntrolador asociado a cada pagina que se incremente cada vez que se hace una referenccia a la pagina. Se selecciona la pagina ocn el contador con enor valor.

Algoritmo de reloj: Modificacion del algoritmo FIFO. Se examina el bit de referencia de la pagina mas antigua (la primera en la lista). Si no esta actuvo se usa para la pagina de reemplazo. Si estuviese activo se le da una

segunda oportunidad a la pagina y se coloca al final de la lista, actualizando su bit de referencia a 0. El algoritmo continúa avanzando al siguiente marco en el orden hasta encontrar una página con el bit de referencia en 0, que será reemplazada.

Si estuviesen todos activados → algoritmo FIFO puro

4.3 Comportamiento de los programas

Viene definido por la secuencia de referencias a pagina que realiza el proceso. Esto es importante para maximizar el rendimiento del sistema de memoria virtual.

Tipos de propiedad de localidad

- **Temporal:** posición de memoria referenciada recientemente. Probable que vuelva a ser referenciada
- **Espacial:** si cierta pocicion de memoria ha sido referenciada es altamente probable que las adyacentes tambien lo sean.

Conjunto de trabajo

El **conjunto de trabajo** de un proceso es el conjunto de paginas que son referencicadas frecuentemente en un determinado intervalo de tiempo.

Observaciones:

- Mientras el conjunto de paginas pueda residir en MP, el nº de faltas no crece mucho
- Si eliminados de MP páginas de ese conjunto, la activacion de paginacion crece mucho

Propiedades:

- Los ocnjuntos de trabajo son transitorios
- No se puede predecir el tamaño futuro de un conjunto de trabajo
- Difieren unos de otros sustancialmente

Teoria del conjunto de trabajo

Un proceso solo puede ejecutarse si su conjunto de trabajo esta en memoria principal.

Una pagina no puede retirarse de memroia principal si esta dentro del conjunto de trabajo del proceso en ejecución.

4.4 Hiperpaginación

Se produce cuando el numero de marcos de pagina asignado a un proceso no es suficiente para almacenar las paginas rederenciadas activamente por el mismo, se produce un numero elevado de fallos de pagina. El proceso pasa mas tiempo en la cola del servicio del dispositivo de paginación que ejecutandose.

Tipos de asginacioón:

- **Asignacion fija:** el nº de marcas de pagina asignado al proceso no es suficiente. Se produce hiperpaginacion y el proceso aumenta

considerablebente su tiempo de ejecucion, el resto de procesos no se ve afectado.

- **Asignación dinamica:** el nº de marcos de un proceso se va adaptando segun necesidad pero si el nº de marcos en el sistema no es suficiente se produciran fallos de pagina frecuentes y el sistema sufrira hiperpaginacion. Se dedica el tiempo a tratar los fallos de pagina.

  Solucion: suspender uno o varios procesos libreando sus paginas.

Es necesario establecer una estrategia de control de carfa que ajuste el grado de multiprogramacion en e sistema para evitar la hiperpaginación.

4.5 Algoritmos de asignación y sustitución

Algoritmo basa en el modelo de WS

Es una técnica utilizada en sistemas operativos para gestionar la memoria virtual. Su objetivo principal es determinar qué páginas de un proceso deben permanecer en memoria para minimizar los fallos de página y optimizar el uso de los marcos disponibles.

Para evitar la hiperpaginacion hay que determinar los conjuntos de trabajos entre los procesos activos `WS = pags referenciadas` en el intervalor `(t-V,t]` que se mantienen en la memoria principal.

Conjunto de trabajo: conjunto de paginas signadas por el proceso en las ultimas `n referenciadas`

- *n* → ventana del conjunto de trabajo, es un factor critico
- *n* muy pequeña → la ventana podria no englobat la situacion actual del proceso.
- *n* muy grande → la ventana engloba varias fases de ejecución.

El SO es capaz de detectar cual es el conjunto de trabajo de cada proceso.

Estategia de asginación dinamica con reeemplzado local y control de carga.

- Conjunto d etrabajo decrece si librean marcos
- Conjunto de trabajo crece se asignan marcos. Si no hay marcos libres hay que realizar un control de carga suspendiendo uno o mas procesos y liberando sus paginas

Algoritmo FFP (Frecuencia de falta de pagina)

Se basa en controlar la frecuencia de fallos de pagina de cada proceso.

- Si la `frecuencia de fallos > limite de fallos` se asignan paginas adicionales al proceso. Si es necesario se liberan paginas.
- Si la `frecuencia de fallo < limite de fallos,` se liberan marcos a traves de un algoritmo de reemplazo.

`Intervalo entre 2 faltas de paginas = tiempo actual - tiempo de la falta de pagina anterior`