Gestión de memoria en Servidores de Información Multimedia

Servidores de Información Multimedia

2º Ingeniero de Telecomunicación (Esp. Sonido e Imagen)

Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid

2 Índice

- Objetivos del sistema de gestión de memoria
- Modelo de memoria de un proceso
- Esquemas de memoria basados en asignación contigua
- Intercambio o swapping
- Memoria virtual

3 Bibliografía

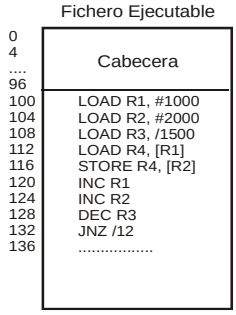
- A. Silberschatz, P. B. Galvin. sistema Operativos. 5ª ed.
- W. Stallings. sistema Operativos, 4ª ed.

1. Objetivos del sistema de gestión de memoria

- En sistemas con multiproceso, el S.O. debe "repartir" los recursos entre los procesos existentes:
  - Reparto de procesador: **Gestión de procesos**
  - Reparto de memoria: **Gestión de memoria**
- Objetivos del Gestor de Memoria
  A. Espacios lógicos independientes
  B. Protección entre procesos
  C. Compartición de Memoria (procesos ligeros)
  D. Soporte a las regiones del proceso
  E. Maximizar el grado de multiprogramación
  F. Mapas de memoria de un tamaño adecuado (normalmente grandes)

A. Espacios lógicos independientes

- A priori no se conoce la **posición** de memoria que **ocupará** un programa cuando vaya a ejecutarse (estado de ocupación de la memoria)
- Código en ejecutable genera referencias entre 0 y N
- Ejemplo: Programa que copia un vector



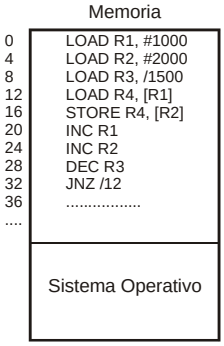
–Vector destino a partir de dirección 2000

–Tamaño del vector en dirección 1500

–Vector origen a partir de dirección 1000

1. Objetivos del sistema de gestión de memoria

- Se supone que el código del SO reside en las posiciones más altas.
- El programa se carga en la posición 0 y para que se ejecute ha de pasársele el control (es decir, que el contador del programa apunte a esta posición).
- **Problema**:Como se ve, no coinciden las direcciones usadas en el programa con la posición a partir de la cual se carga el programa. Suponer por ejemplo, que el programa se cargara a partir de la posición 10000....
- **Solución**: **Reubicación** de direcciones lógicas a físicas



1. Objetivos del sistema de gestión de memoria

- **Reubicación**: Traducir direcciones lógicas a direcciones físicas dependientes del hardware mediante una función de traducción. La reubicación permite crear un espacio lógico independiente para cada proceso y el S.O. debe poder acceder a los espacios lógicos para realizar la traducción.
  - Direcciones lógicas: direcciones de memoria generadas por el programa
  - Direcciones físicas: direcciones de memoria principal asignadas
- **Función de traducción**:

  Traducción(IdProc, dir_lógica) $\rightarrow$ *dir_física*

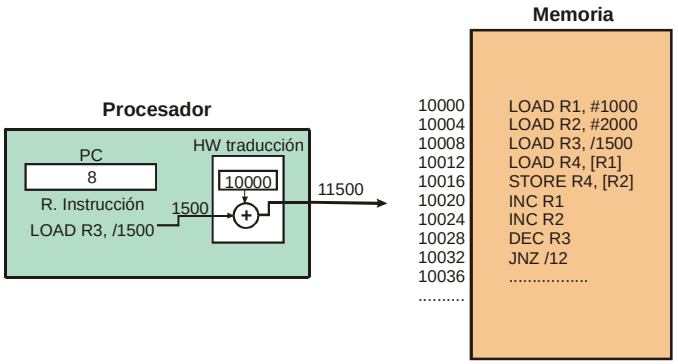
  - Ejemplo:

    El programa tiene asignada memoria a partir de la dirección 10000

    Reubicación: sumar 10000 a direcciones lógicas
  - Implementación: Hardware o Software

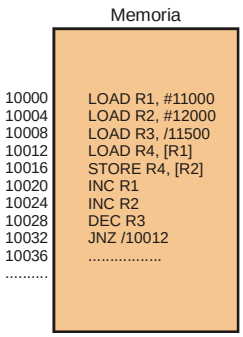
1. Objetivos del sistema de gestión de memoria

- **Reubicación Hardware**: la MMU (memory management unit) se encarga de la traducción
- **Proceso**:
  - Programa se carga en memoria sin modificar
  - El S.O. almacena por cada proceso su función de traducción
  - El S.O. especifica a la MMU qué función aplicar para cada proceso



1. Objetivos del sistema de gestión de memoria

- **Reubicación software**: traducción de direcciones durante carga del programa. Esta solución se usa en sistemas sin el hardware específico de traducción (MMU).
- **Proceso**:
  - El programa se carga con las direcciones ya traducidas
  - Se genera un código diferente del programa ejecutable
- **Desventajas**:
  - No asegura protección (no se verifica cada dirección a usar, sino que se usan las direcciones generadas tras el proceso de carga)
  - No permite mover programa en tiempo de ejecución (suponer que es necesaria la reubicación del espacio asignado al proceso, por necesitar más espacio.....)



B.Protección entre procesos

La protección es diferente según sea un sistema Mono o Multi programado:

Sistema MonoProgramado: La intrusión se dará solamente entre el programa y el S.O.

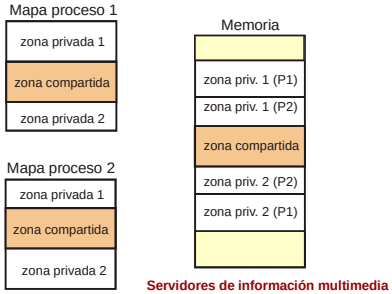
Sistema MultiProgramado: La intrusión puede venir tanto de otros procesos como de otros usuarios además del riesgo del sistema Monoprogramado.

- **Acciones**
  - La traducción de direcciones debe crear espacios disjuntos
  - Es necesario validar todas las direcciones que genera el programa
    - La detección de alguna intrusión debe realizarla el hardware del procesador (MMU), ya que hay que realizarla en tiempo de ejecución
    - El tratamiento de alguna intrusión lo hace el SO
  - En sistemas con mapa de E/S y memoria común:
    - Traducción permite impedir que los procesos accedan directamente a dispositivos de E/S

C.Compartición de Memoria (procesos ligeros)

La compartición de memoria entre procesos da soporte a la creación de procesos ligeros y está controlado por el S.O.

- **Acciones**:
  - Las direcciones lógicas de 2 o más procesos se corresponderán con una misma dirección física.
  - La memoria asignada a cada proceso no puede ser ya contigua.
  - La función de traducción en estos casos se va haciendo más compleja.
- **Ventajas**:
  - Procesos ejecutando mismo programa comparten su código
  - Mecanismo de comunicación entre procesos muy rápido



D. Soporte de las regiones del proceso

El mapa de memoria de un proceso no es homogéneo, ya que las regiones contienen diferentes tipos de información (código, datos y pila normalmente) y poseen diferentes características

- **Acciones**:
  - Hacer mapa de memoria dinámico:
    - Regiones cambian de tamaño (p.ej. pila)
    - Se crean y destruyen regiones (el sistema de memoria debe controlar qué regiones están presentes, así como su tamaño)
    - Existen zonas sin asignar (huecos, con carácter dinámico)
  - Detectar accesos no permitidos a una región (de sólo lectura por ej)
  - Detectar accesos a huecos
  - Evitar reservar espacio para huecos
  - Guardar y gestionar una tabla de regiones para cada proceso

E. Maximizar el grado de multiprogramación

El reparto de memoria debe ser tal que maximice el grado de multiprogramación para evitar el desperdicio de memoria

- **Memoria desperdiciada**
  - Restos (huecos) inutilizables (fragmentación)
  - Tablas requeridas por gestor de memoria
- **Acciones**
  - Crear bloques de asignación de menor tamaño
  - Gestionar las tablas de asignación más eficientemente
  - Se opta por la paginación
  - Uso de memoria virtual para aumentar grado de multiprogramación

Aprovechamiento de memoria óptimo es irrealizable!!!! Tablas de gestión demasiado grandes

Memoria

0	Dirección 50 del proceso 4
1	Dirección 10 del proceso 6
2	Dirección 95 del proceso 7
3	Dirección 56 del proceso 8
4	Dirección 0 del proceso 12
5	Dirección 5 del proceso 20
6	Dirección 0 del proceso 1
	....
N-1	Dirección 88 del proceso 9
N	Dirección 51 del proceso 4

F. Mapas de memoria de un tamaño adecuado (normalmente grandes)

Los procesos necesitan cada vez mapas más grandes: aplicaciones más novedosas, más recursos gráficos, más carga computacional....

- **Acciones**
  - Utilizar Memoria Virtual
  - Hacer que el usuario disponga virtualmente de una enorme cantidad de memoria física
- **Otras opciones (antigua)**
  - *Overlays*
    - Programa dividido en fases que se ejecutan sucesivamente
    - En cada momento sólo hay una fase residente en memoria
    - Cada fase realiza su labor y carga la siguiente
    - No es transparente: Toda la labor realizada por programador

2. Modelo de memoria de un proceso

17 2.1. Introducción

- El S.O. es el responsable de la gestión de memoria de cualquier proceso.
- El mapa inicial de memoria de un proceso está relacionado con el archivo ejecutable que generará el proceso
- ¿Cómo generar el mapa de memoria inicial a partir del ejecutable?
- ¿Cómo se organiza el mapa de memoria?
- ¿Cuáles son sus características básicas?
- ¿Qué operaciones se pueden realizar en el mapa de memoria?

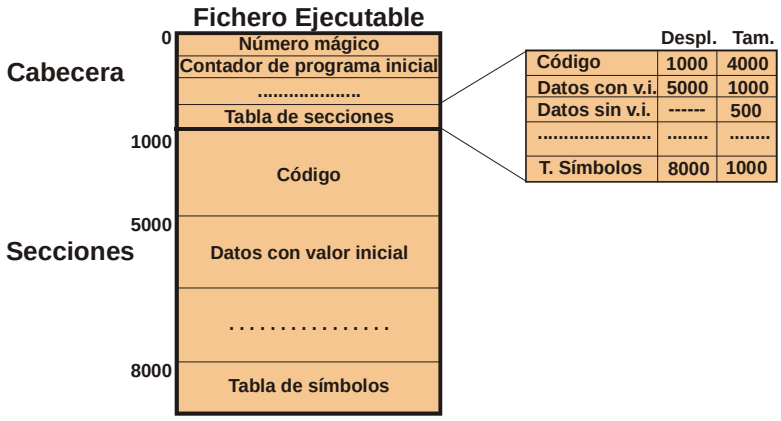
18 2.2. Fases en la generación de un ejecutable

Ficheros ejecutables

- Distintos fabricantes usan diferentes formatos:
  - Ejemplo: En Linux *Executable and Linkable Format* (ELF)
- Estructura: Cabecera y conjunto de secciones
- **Cabecera**: información de control que permite interpretar el contenido del ejecutable. Suele incluir la siguiente información:
  - **Número mágico** que identifica a ejecutable. Por ejemplo, en formato ELF el primer byte debe contener el valor hexadecimal 7f, los tres siguientes los caracteres E, L y F
  - **Punto de entrada** del programa: es decir, el valor que inicialmente contendrá el contador del programa
  - **Tabla de secciones**. Para cada una de ellas se especifica: tipo, dirección de comienzo en el archivo y tamaño.

    Ejemplo:
    - » Tabla de símbolos para depuración
    - » Sección de código
    - » Sección de datos

2.2. Fases en la generación de un ejecutable



2.2. Fases en la generación de un ejecutable

- Las **secciones más relevantes** se volcarán en el mapa de memoria del proceso: código, datos con valor inicial y datos sin valor inicial
  - **Código** (texto): contiene el código del programa
  - **Datos con valor inicial**: variables globales inicializadas
  - **Datos sin valor inicial**: variables globales no inicializadas: aunque aparece en la tabla de secciones no se almacena en el ejecutable, ya que su contenido no es relevante
- No hay sección vinculada a variables locales, ya que estas tienen carácter dinámico (se crearán en la pila del proceso, cuando los procedimientos en que aparecen sean invocados)

21 2.2. Fases en la generación de un ejecutable

- **Variables globales**
  - Estáticas
  - Se crean al iniciarse programa
  - Existen durante toda la ejecución del proceso
  - Dirección fija en memoria y en ejecutable

- **Variables locales y parámetros**
  - Dinámicas
  - Almacenadas en la pila
  - Se crean al invocar la función
  - Se destruyen al retornar
  - La dirección se calcula en tiempo de ejecución
  - Recursividad: varias instancias de una variable

22 2.3. Mapa de memoria de un proceso

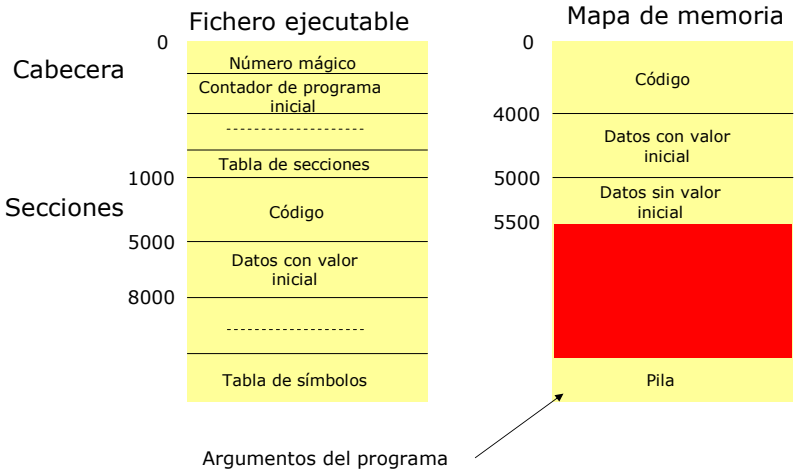
- El mapa de memoria o imagen del proceso estará compuesto por **un conjunto de regiones o segmentos**; cada una de ellas almacena cierto tipo de información

- **Cada región**:
  - Tiene asociada una información (un "objeto de memoria")
  - Consiste en una zona contigua tratada como unidad al proteger o compartir
  - Se caracteriza por:
    - Dirección de comienzo y tamaño inicial
    - Soporte: donde se almacena su contenido inicial (soporte en archivo y sin soporte, es decir, objeto sin contenido inicial)
    - Protección: RWX
    - Uso compartido o privado
    - Tamaño fijo o variable

23 2.3. Mapa de memoria de un proceso

- La **ejecución de un programa crea un mapa de memoria** a partir del archivo ejecutable. Cada sección del ejecutable da lugar a una región del mapa inicial.
  - Código (texto): compartida, lectura y ejecución, tamaño fijo, soporte en archivo ejecutable
  - Datos con valor inicial: privada, lectura y escritura, tamaño fijo, soporte en archivo ejecutable
  - Datos sin valor inicial: privada, lectura y escritura, tamaño fijo, sin soporte (se rellena a ceros en algunos lenguajes)
  - Pila: privada, lectura y escritura, tamaño variable, sin soporte. Crece hacia direcciones más bajas. La pila inicial sólo contiene los argumentos de llamada al programa

24 2.3. Mapa de memoria de un proceso



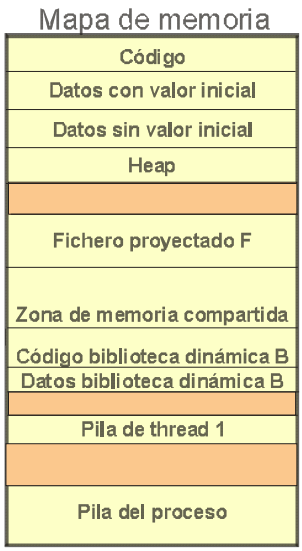
25 2.3. Mapa de memoria de un proceso

- **Durante ejecución** de proceso se **crean nuevas regiones**. Es decir, el mapa de memoria tiene un carácter dinámico. Las nuevas regiones creadas en tiempo de ejecución pueden ser:
  - Región de Heap
    - Soporte de memoria dinámica (`malloc` en C)
    - Privada, lectura y escritura, tamaño variable, sin soporte (inicializada a cero)
    - Crece hacia direcciones más altas
  - Archivo proyectado
    - Región asociada al archivo proyectado
    - Tamaño variable, soporte en archivo
    - Protección y carácter compartido o privado especificado en la proyección
  - Memoria compartida
    - Región asociada a la zona de memoria compartida
    - Compartida, tamaño variable, sin soporte (inicializada a 0)
    - Protección especificada en proyección
  - Pilas de threads
    - Cada pila de thread corresponde con una región.
    - Estas regiones constan de las mismas características que las asociadas a la pila del proceso

26 2.3. Mapa de memoria de un proceso

Región	Soporte	Protección	Comp/Priv	Tamaño
Código	Fichero	RX	Compartida	Fijo
Dat. con v.i.	Fichero	RW	Privada	Fijo
Dat. sin v.i.	Sin soporte	RW	Privada	Fijo
Pilas	Sin soporte	RW	Privada	Variable
Heap	Sin soporte	RW	Privada	Variable
F. Proyect.	Fichero	por usuario	Comp./Priv.	Variable
M. Comp.	Sin soporte	por usuario	Compartida	Variable

27 2.3. Mapa de memoria de un proceso



28 2.4. Operaciones sobre regiones

Para estudiar la evolución del mapa de memoria a lo largo de la ejecución de un proceso, se pueden distinguir las siguientes operaciones:

- Crear región: Implícitamente al crear mapa inicial (por parte del SO) o por solicitud del programa en tiempo de ejecución de ejecución (por ejemplo, al cargar una biblioteca dinámica)
- Eliminar región: Implícitamente al terminar el proceso o por solicitud del programa en tiempo de ejecución (por ejemplo, al desproyectar un archivo)
- Cambiar tamaño de la región: Implícitamente para la pila o por solicitud del programa para el heap (cuando se hace malloc)
- Duplicar región: Operación requerida por el servicio fork de POSIX

3. Esquemas de memoria basados en asignación contigua

3.1. Esquema hardware
3.2. Gestión del SO
3.3. Política de asignación de espacio
3.4. Valoración del esquema contiguo

30 3.1. Esquema Hardware

El mapa de proceso se ubica en una zona contigua de la memoria principal.

Proceso:

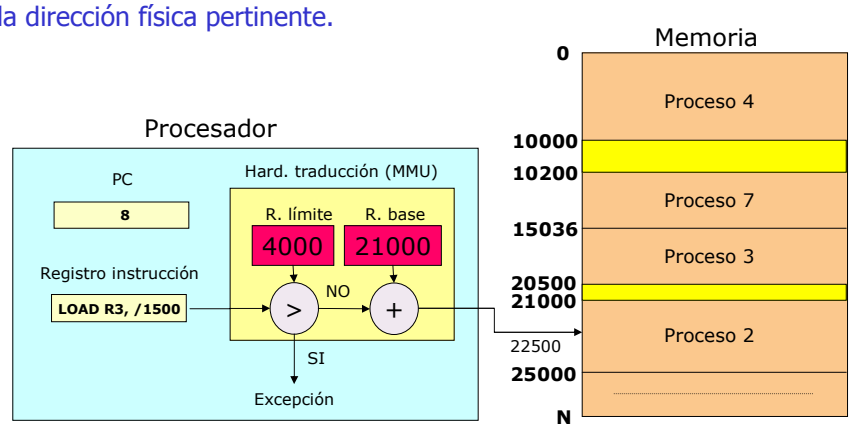
- El S.O. **busca un hueco** en memoria de tamaño suficiente para alojar su mapa de memoria del proceso que comienza.
- El S.O. **reserva la parte** del hueco **necesaria** y crea en ella el mapa inicial del proceso
- Se establece la **función de traducción**, de forma que las direcciones del programa se correspondan con las direcciones existentes en el hueco asignado.

Hardware requerido:

- **Registros valla** (registro base y registro límite). Estos dos registros sólo son accesibles en modo privilegiado.
- Los registros valla están desocupados cuando el S.O. toma el control para acceder a todo el mapa de memoria.

31 3.1. Esquema Hardware

- **Registro límite**: Se comprobará que las direcciones usadas por el proceso no excedan el valor almacenado en él.
- **Registro base**: Una vez realizada la comprobación anterior, se suma a cada dirección el valor contenido en este registro, de forma que se obtiene la dirección física pertinente.



32 3.2. Gestión del SO

- **El S.O. almacena en el BCP (o PCB) cuáles son los valores de los registros valla.**
- Dedica **una estructura** para conocer en todo momento el estado de la memoria, identificando qué huecos están libres. Normalmente se usa una lista en la que se almacena la dirección inicial y el tamaño de cada hueco.
- La gestión de esta lista **obliga a comprobar**, al desocupar espacio, si el nuevo espacio libre puede unirse a huecos vecinos.
- Problema:Según se van ejecutando procesos van quedando fragmentos de memoria libres, que debido a su tamaño no podrán ser usados en asignaciones de espacio a memoria. Este problema se denomina **fragmentación externa** y conlleva una mala gestión de memoria.
- Solución tradicional: Compactar los huecos de forma que queden contiguos. Para ello es necesario **reajustar los registros valla** de los procesos. INEFICIENTE.

33 3.3. Política de asignación de espacio

- El S.O. debe considerar qué espacio, de los huecos libres, se usará intentando encontrar un equilibrio entre buen aprovechamiento de espacio y tiempo de respuesta corto, es decir, se aplica un algoritmo de decisión que debe ser eficiente.
- Este problema es un clásico: ¿Cómo asignar espacio para su aprovechamiento óptimo?. Existen tres posibles enfoques:
  - Mejor ajuste (best-fit). Se elige la zona libre más pequeña donde quepa el mapa del proceso.
    - Problema: Conlleva crear nuevos huecos de tamaño pequeño. Además, elegir el hueco más pequeño obliga a mantener ordenados por tamaño los huecos disponibles. No es eficiente
  - b) Peor ajuste. Se busca el hueco más grande, intentando evitar la generación de huecos pequeños. Sigue precisando mantener el control de los tamaños
  - c) El primero que ajuste (first-fit). Suele ser la mejor política. Muy eficiente, ya que basta con encontrar una zona libre de tamaño suficiente, y ofrece un aprovechamiento aceptable

34 3.4. Valoración del esquema contiguo

- Valoracion:
  - Espacios independientes para procesos: mediante registros valla
  - Protección: mediante registros valla
  - Compartir memoria: no es posible
  - Soporte de regiones:
    - No existe (no hay mecanismo de permisos sobre el espacio asignado a cada proceso)
    - Se reserva espacio para huecos, ya que el espacio asignado al proceso en primera instancia debe servir para todo su tiempo de vida
  - Maximizar rendimiento:
    - Mal aprovechamiento de memoria por fragmentación externa
  - Mapas de MV de tamaño adecuado:
    - No permite memoria virtual

4. Intercambio

Intercambio

¿Qué hacer si no caben todos los programas en memoria principal?

Usar disco (dispositivo ***swap***) como respaldo de la memoria principal. Si no caben en memoria todos los procesos activos, se elige un proceso residente y se copia en disco su imagen de memoria

Suspender (*swap-out*) a los procesos bloqueados. El proceso de suspensión no implica copiar toda la imagen del proceso (por ejemplo, no es preciso ocupar los huecos ni el código, al poder recuperarse fácilmente del ejecutable). Un proceso suspendido vuelve a cargarse (***swap-in***) cuando esté listo para ejecutar y haya espacio en memoria.

Políticas de asignación de espacio en swap:

Preasignación: al crear el proceso se reserva espacio de swap.

NO Preasignación: sólo se reserva espacio de swap al suspender.

5. Memoria virtual

- La técnica de la MV se usa prácticamente en todos los SSOO modernos. Esta técnica se basa en **transferir información entre memoria principal y memoria secundaria** (por lo que involucra varios niveles de la jerarquía de memoria)
- Suele implementarse en un **esquema de paginación** (es decir, la unidad de información intercambiada entre los diferentes niveles de la jerarquía de memoria es la página)
- Elemento Clave: **Proximidad referencial** habitual de los procesos. Esta propiedad permite que un proceso puede funcionar disponiendo en memoria de una parte de su imagen de memoria (conjunto residente).
- Objetivo final: conseguir que la información necesaria para un proceso (conjunto de trabajo) vaya ocupando la memoria principal según se va necesitando (es decir, **conjunto de trabajo === conjunto residente**)

39 5.1. Introducción

- Ventajas:
  - Aumento del **grado de multiprogramación**. Por tanto, aumento en el rendimiento del sistema
  - Posibilidad de ejecutar **programas más grandes** que la MV disponible
- El uso de la MV **no implica que se acelere** la ejecución del programa (más bien al contrario, debido a la sobrecarga asociada a los movimientos de información entre niveles de la jerarquía). Este mecanismo no es apropiado para sistemas de tiempo real.

40 5.2. Paginación

- Página: Zona contigua de memoria de determinado tamaño. (Por motivos de eficiencia se suele trabajar siempre con tamaños potencia de 2. Ej:4 KB.)
- Organización:
  - El mapa de memoria del proceso se considera dividido en páginas.
  - La memoria principal se considera dividida en marcos de página (tamaño de marco = tamaño de página).
  - Los marcos contendrán páginas de los procesos en ejecución
  - La tabla de páginas (TP) relaciona cada página con el marco que la contiene. El hardware de traducción (MMU) usa la tabla de páginas para traducir direcciones lógicas a físicas
  - Típicamente la MMU usa dos tablas de páginas (TP):
    - TP del usuario: Por ejemplo, direcciones lógicas que empiezan por 0
    - TP del sistema: Por ejemplo, direcciones lógicas que empiezan por 1. Estas sólo se podrán usar en modo sistema

41 5.2. Paginación

- Cada entrada de la tabla de páginas contendrá, además del número de marco asociado con la página:
  a) **Información de protección**: tipo de acceso permitido RWX
  b) **Bit de página válida/inválida**: para indicar si dicha entrada contiene una traducción asociada, es decir, si se corresponde realmente con un marco
  c) **Bit de página accedida (*Ref*)**: activado cuando se accede
  d) **Bit de página modificada (*Mod*)**: activado cuando se escribe
  e) **Bit de desactivación de caché**: se usa cuando la entrada corresponde con direcciones de E/S

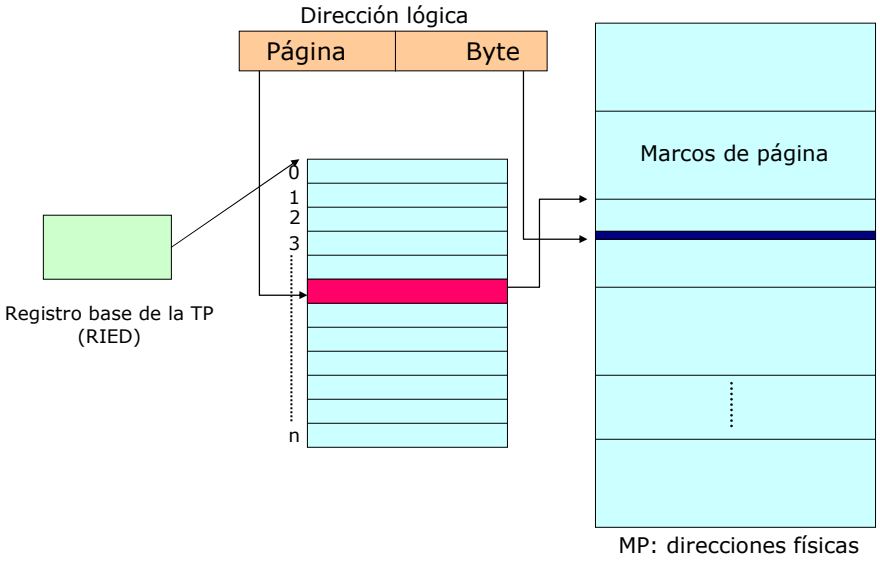
- Tamaño de página: La **elección de tamaño de página** está condicionada por diversos factores, entre los que hay que conseguir equilibrio:
  - Potencia de 2 y múltiplo del tamaño del bloque de disco
  - mejor pequeño por:
    - Menor fragmentación
    - Se ajusta mejor al conjunto de trabajo
  - mejor grande por:
    - Tablas más pequeñas
    - Mejor rendimiento de dispositivos de E/S
  - Compromiso (entre 2K y 16K)

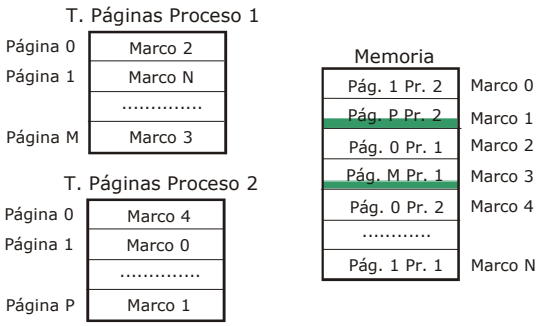
42 5.2. Paginación

- Dirección:Una dirección lógica se obtiene a partir de:

  nº página + desplazamiento

- Problema:
  - La paginación **no ofrece una solución óptima**. Lo **ideal** sería que cada palabra del mapa de memoria de un proceso pudiera ubicarse en cualquier dirección. Esta solución es inviable debido al coste de traducción
  - Al no ser así, con la paginación se asigna a **cada proceso un número entero de marcos** de página, aunque el espacio de su mapa de memoria no sea un múltiplo entero del tamaño de página. Por tanto, se irán generando huecos (memoria no aprovechada, en la última de las páginas que no se llena de forma completa). Este proceso se denomina **fragmentación interna** (implica que, en término medio, cada proceso desperdicia la mitad de una página)





- Problema: Fragmentación, la memoria asignada es mayor que la memoria requerida y por lo tanto, se desperdicia cierta cantidad de espacio

45 5.2. Paginación

- Otras cuestiones:
  - En este esquema, el S.O. mantiene una **tabla de páginas por cada proceso**. Cuando se produce un cambio de contexto se indica a la MMU qué tabla de páginas usar
  - El S.O. mantiene **una única tabla de páginas para sí mismo**. Así, todos los procesos comparten el SO. Cuando un proceso se ejecuta en modo sistema accede directamente a su mapa y al del SO
  - **S.O. mantiene tabla de marcos**, como forma de mantener información de estado de la memoria principal. De cada marco se conoce su estado (libre, ocupado, etc)
  - S.O. mantiene **tabla de regiones por cada proceso**, indicando las características de cada región y qué rango de páginas pertenecen a cada región
- Desperdicio de espacio**: Mucho mayor gasto en tablas que con asignación contigua**: es el precio de mucha mayor funcionalidad

46 5.2. Paginación

- Implementación de la tabla de páginas:
  - Las tablas de páginas se mantiene **normalmente en memoria principal**. Problemas: eficiencia y gasto de almacenamiento
    - **a) Eficiencia**: cada acceso lógico requiere dos accesos a memoria principal, a la tabla de páginas + al propio dato o instrucción. Solución: *caché* de traducciones –> TLB
    - **b) Gasto de almacenamiento**: tablas muy grandes. Ejemplo: páginas 4K, dir. lógica 32 bits y 4 bytes por entrada, la tabla de páginas de cada proceso tendrá 4MB. Solución: ***tablas multinivel*** y ***tablas invertidas***
- Valoración :
  - Espacios independientes para procesos: mediante tablas de páginas
  - Protección: mediante tablas de páginas
  - Compartir memoria: entradas corresponden con mismo marco (bajo supervisión del SO dos procesos pueden compartir una página asociada al mismo marco)
  - Soporte de regiones: bits de protección, bit de validez: **no se reserva espacio para huecos**
  - Maximizar rendimiento: Si, al permitir esquemas de memoria más flexibles
  - Mapas de tamaño adecuado: sí, al permitir esquemas de memoria virtual

47 5.2. Paginación

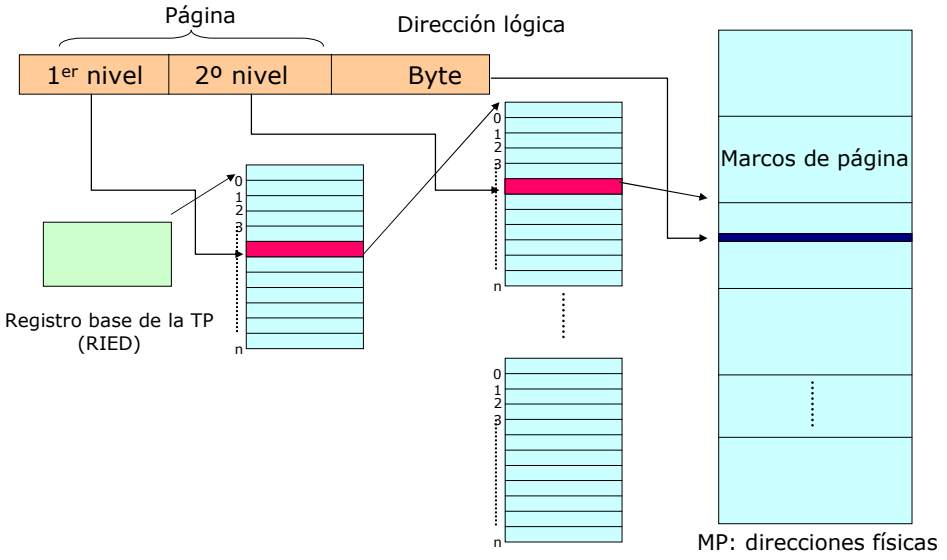
TLB (Translation Look-aside Buffer): Consta de una **memoria asociativa** con información sobre últimas páginas accedidas.

- Para el multiproceso, existen varias posibilidades:
  - La TLB **no incluye información** del proceso. En este caso, habrá que invalidar la TLB en los cambios de contexto
  - Entradas en TLB **incluyen información** sobre proceso: en este caso, debe existir un registro de UCP para mantener la identificación del proceso actual
- Implementación para resolver los fallos de TLB
  - **HW**: La MMU consulta la TLB y si falla, se usa la TP en memoria.
    - Ventajas: Es un proceso muy rápido
    - Inconvenientes: Actualizar la TP en cambios de contexto y hay que invalidar la TLB cuando se produce un cambio de contexto (si no tiene información del PID)
  - **SW:** La MMU no usa la TP, sino que sólo consulta TLB. En caso de fallo, se activa el SO, que ha de buscar la entrada en la tabla de páginas (mediante programa), e insertar en la TLB la traducción hecha, de forma que se pueda reutilizar.
    - Ventajas: flexibilidad, ya que la tabla de páginas puede ser definida a conveniencia, sin restricciones impuestas por el hardware
    - Inconvenientes: Es menos eficiente, ya que parte del proceso de traducción se realiza mediante programa.

48 5.2. Paginación

- Una de las opciones disponibles para disminuir el tamaño requerido por las tablas de páginas es la ***tabla de páginas multinivel***
- Organización:Se trata de una tabla de páginas organizadas en M niveles:
  - Entrada de TP de nivel K apunta a TP de nivel K+1
  - Entrada de último nivel apunta a marco de página
- Dirección: La dirección lógica especifica la entrada a usar en cada nivel
  - 1 campo por nivel + desplazamiento
- Accesos:Un acceso lógico supone M + 1 accesos a memoria. Solución: uso de TLB
- Invaliadación: Si todas las entradas de una TP son inválidas, no se almacena esa TP y se pone inválida la entrada correspondiente de la TP superior
- Ahorro de espacio: Si el proceso usa una parte pequeña de su espacio lógico, se consigue ahorro en espacio para almacenar TPs

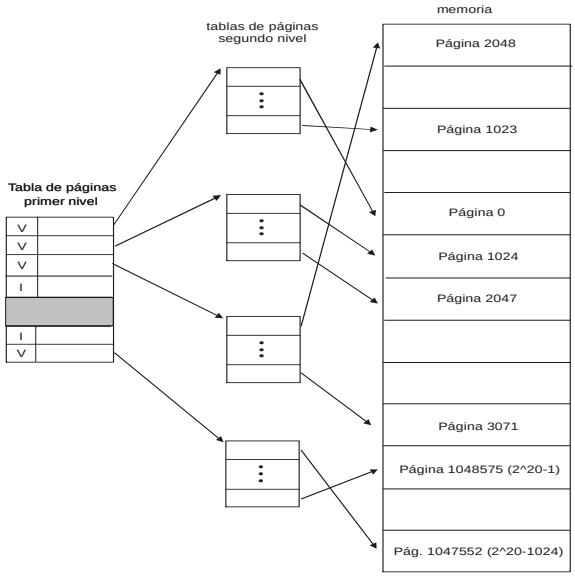
5.2. Paginación



5.2. Paginación

- Ejemplo: Proceso que usa 12MB superiores y 4MB inferiores
  - 2 niveles, páginas de 4K, dir. lógica 32 bits (10 bits por nivel) y 4 bytes por entrada
  - Tamaño: 1 TP N_1 + 4 TP N_2= 5 * 4KB = 20KB (frente a 4MB)
- Ventajas adicionales: permite compartir TPs intermedias y sólo se requiere que esté en memoria la TP de nivel superior. Las restantes pueden estar en disco y traerse por demanda

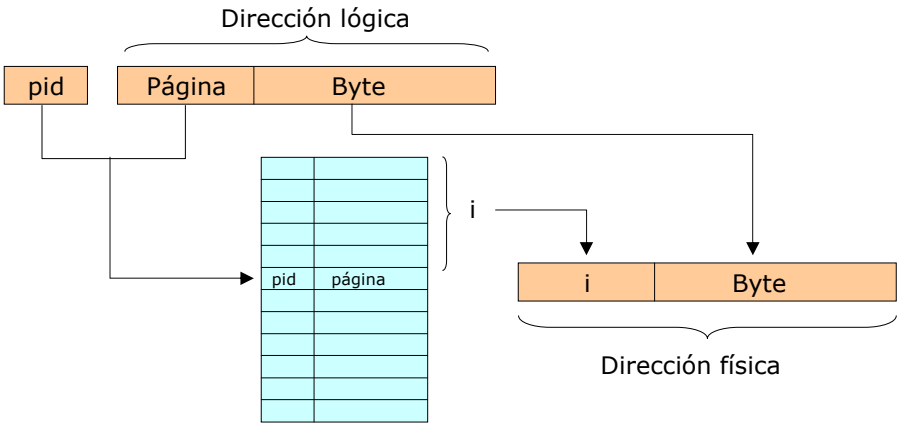
5.2. Paginación



5.2. Paginación

- Otra alternativa para disminuir el espacio necesario por la tabla de páginas es la ***tabla de páginas invertida***
- Organización:
  - La tabla contendrá tantas entradas como marcos de página haya.
  - Cada entrada almacena la página cargada en dicho marco junto con sus características:
    - Número de página,
    - Proceso dueño de la página
  - El tamaño de la tabla de páginas es proporcional a la memoria disponible.
- Accesos:La MMU usa una TLB convencional, pero si falla la traducción se accede a la tabla de páginas invertida. Al estar la tabla organizada por marcos no se puede hacer una búsqueda directa. Debería accederse a todas las entradas en busca de la página correspondiente. Por esta razón se suele organizar como una **tabla hash**
- Ahorro de espacio: Se reduce el espacio de almacenamiento necesario, ya que sólo se guarda información sobre las páginas válidas

53 5.2. Paginación



54 5.3. Segmentación

Con la paginación la MMU no dispone de información sobre las **regiones de los procesos** y sólo entiende de páginas. Por esta razón, el SO debe mantener una **lista de las páginas que componen cada región**, lo que supone varias desventajas:

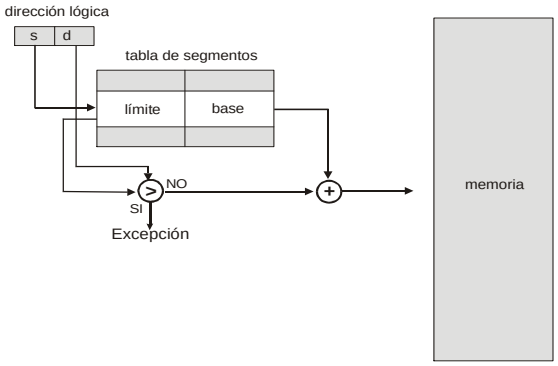
- Al **crear una región** hay que cuidar que todas las páginas asociadas tengan la misma información de control
- Para poder **compartir** una región es preciso que las entradas de varios procesos apunten a los mismos marcos

Segmentación: es un esquema HW que intenta dar soporte directo a las regiones. En él se considera el mapa de memoria como compuesto por varios segmentos.

Se puede considerar que consisten en una **generalización de los registros valla base y límite**, pero usando un par de registros por cada segmento

55 5.3. Segmentación

- Dirección:Una dirección lógica estará compuesta por un número de segmento y un desplazamiento en el mismo.
- Traducción: La forma de realizar la traducción puede apreciarse en la imagen siguiente:



56 5.3. Segmentación

- Tabla de segmentos:En este caso, la MMU usa una tabla de segmentos (TS).
- Organización:
  - El SO mantiene una TS por proceso, de forma que en cada cambio de contexto se notifica a MMU cuál debe usar
  - Cada entrada de TS contiene (entre otros):
    - Registro base y límite del segmento
    - protección: RWX
- Problema: **Fragmentación Externa**
  - En este esquema se produce fragmentación externa, ya que el almacenamiento de los segmentos se realiza de **forma contigua**.
  - El SO debe mantener una lista que le permita conocer qué zonas de memoria están libres y cuáles ocupadas: es decir, **estructuras de datos que identifiquen huecos y zonas asignadas**

57 5.3. Segmentación

- Valoración:
  - Espacios independientes para procesos: mediante su propia TS, que crea un espacio lógico independiente
  - Protección: mediante TS, ofreciendo espacios disjuntos de memoria
  - Compartir memoria: bajo control del SO es posible que dos o más procesos tengan un segmento asociado a la misma zona de memoria
  - Soporte de regiones: bits de protección
  - Maximizar rendimiento : No lo maximiza, por la fragmentación externa
  - Mapas de tamaño adecuado: No cumple este objetivo porque no permite implementar eficientemente un sistema de memoria virtual

Por tanto, tal y como se ha presentado se usa en **muy pocos SO reales**

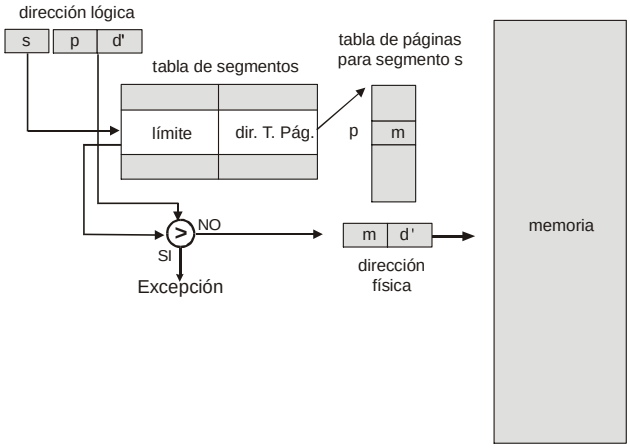
58 5.4. Segmentación paginada

- Se intenta **aunar las ventajas de ambos esquemas**: segmentación y paginación:
  - Segmentación: soporte para regiones.
  - Paginación: mejor uso del espacio de memoria.

- Organización:
  - Entrada en TS apunta a una TP para el segmento.
  - El espacio del segmento está compuesto de varias páginas, de forma que su espacio no tiene que ser contiguo.

5.4. Segmentación paginada

•**Traducción:** La traducción se lleva a cabo según se indica**.**



5.4. Segmentación paginada

- Valoración:
  - Espacios independientes para procesos: mediante TS
  - Protección: mediante TS
  - Compartir memoria: bajo control del SO, podemos hacer que una misma entrada aparezca en diferentes TS (es decir, pueda ser usado por varios procesos)
  - Soporte de regiones: bits de protección
  - Maximizar rendimiento: la paginación aprovecha eficientemente el espacio de memoria.
  - Mapas de tamaño adecuado: permite esquemas de memoria virtual
- Ventajas: Frente a paginación sin segmentos facilita al SO la gestión de las regiones, pero requiere HW más complejo

61 5.5. Paginación por demanda

- Una vez analizados los diferentes esquemas hardware vamos a ver cómo se articulan para construir un esquema de memoria virtual (estos esquemas también pueden usarse sin memoria virtual, pero en la actualidad su uso está siempre vinculado a la memoria virtual)

- Normalmente la memoria virtual se construye sobre esquemas de paginación pura o segmentada. De esta forma, la unidad de información intercambiada entre memoria principal y secundaria es la página.

- Normalmente la transferencia de información se lleva a cabo bajo demanda (***paginación por demanda***). De esta forma, cuando un proceso necesita acceder a una página que no está en memoria principal, se genera un **fallo de página** y el SO se encarga de transferirla desde la memoria secundaria. Si al traerla no hay espacio suficiente en MP, será necesario desalojar alguna de las páginas actuales (ello se hace mediante un ***algoritmo de reemplazo***)

62 5.5. Paginación por demanda

- La construcción de un sistema de memoria virtual sobre un procesador con paginación implica usar un **bit de validez en las entradas de la tabla de páginas**, que indica si la página es válida. Estarán marcadas como inválidas todas las páginas que no residen en MP, así como las que constituyen huecos en el mapa de memoria.

- Para las entradas correspondientes a páginas no residentes en MP, la entrada principal, en lugar de almacenar el marco donde reside contendrá la **dirección del dispositivo de memoria en que se encuentra almacenada**. De forma que cuando se produce un acceso a una de estas páginas, se produce una excepción y se activa el SO, responsable de hacer la transferencia desde memoria secundaria.

- Algunos sistemas también usan la técnica de ***prepaginación***. Al ocurrir un fallo de página no sólo traen la página en cuestión, sino también las cercanas, al suponerse que se usarán en un corto plazo de tiempo. La efectividad de esta técnica dependerá del acierto de la predicción.

63 5.5. Paginación por demanda

Veamos cómo se gestiona la ocurrencia de un **fallo de página**:

- La MMU genera una excepción. Normalmente deja en un registro especial la dirección que causó el fallo
- Se activa el SO, que comprueba:
  - Si la página es inválida, se aborta el proceso.
  - Si la página es ausente se siguen los pasos siguientes:
    a) Se consulta la tabla de marcos para ver si hay algún hueco libre:
       i. Si no hay ningún marco libre se aplica el algoritmo de reemplazo, que decidirá el marco a desalojar. La página almacenada hasta entonces se marca como inválida. Si ha sido modificada, antes hay que salvar su contenido en memoria secundaria
       ii. Si hay marco libre: se inicia la lectura desde memoria secundaria y se vuelca al marco, marcándose como válida dicha entrada en la TP y apuntando el marco en que está almacenada

64 5.5. Paginación por demanda

- En el peor de los casos un fallo de página puede suponer dos operaciones de E/S:
  a) Salvaguarda de la página expulsada
  b) Lectura de la página nueva

Dos políticas definen el funcionamiento del sistema de memoria

- Política de reemplazos:
  - Reemplazo local: Sólo puede usarse para reemplazo un marco asignado al proceso que causa fallo
  - Reemplazo global: Puede usarse para reemplazo cualquier marco

- Política de asignación de espacio a los procesos:
  - Asignación fija: El número de marcos de página para cada proceso es fijo.
  - Asignación dinámica: El número de marcos de página para cada proceso es dinámico

65 5.6. Políticas de reemplazo

- Objetivo: Minimizar la tasa de fallos de página.

- Cada algoritmo descrito tiene versión local y global:
  - Local: criterio se aplica a las páginas residentes del proceso
  - Global: criterio se aplica a todas las páginas residentes

- Algoritmos a estudiar
  A. Óptimo
  B. FIFO
  C. Reloj (o segunda oportunidad)
  D. LRU
  E. B*uffering* de páginas
  F. Retención de páginas en memoria

66 A) Algoritmo óptimo

- Criterio: Página residente que tardará más en accederse
- Implementación: Irrealizable, ya que supone disponer de una predicción fiable del uso de las páginas en un futuro a medio plazo
- Versión local y global
- Interés para estudios analíticos comparativos

67 B) Algoritmo FIFO

- Criterio: se elimina la página que lleva más tiempo residente
- Implementación: Fácil
  - Páginas residentes en orden FIFO –› se expulsa la primera
  - No requiere hardware especial
  - En el caso de estrategia local se mantiene una lista de páginas por cada proceso. En el caso global, basta con una única lista
- Problema:
  - Una página que lleva mucho tiempo residente puede seguir accediéndose frecuentemente.
  - Su criterio no se basa en el uso de la página.
  - Anomalía de Belady: Se pueden encontrar ejemplos en que al aumentar el número de marcos aumenta el número de fallos de página

68 C) Algoritmo de segunda oportunidad o del reloj

- Se trata de una modificación del algoritmo FIFO, para evitar que una página residente desde hace tiempo sea desalojada pese a estar siendo usada. Para ello se usa el **bit de referencia** *Ref* de las páginas, con lo que se detecta su uso
- Criterio:
  - Si la página elegida por FIFO no tiene activo el bit *Ref,* es la página expulsada
  - Si lo tiene activo se da 2ª oportunidad antes de expulsar: se desactiva el bit *Ref,* se pone página al final de FIFO, se aplica criterio a la siguiente página
- Implementación: Se puede implementar el orden FIFO mediante una lista circular con una referencia a la primera página de la lista: se visualiza como un reloj donde la referencia a la primera página es la aguja del reloj

69 D) Algoritmo LRU (least recently used)

- Criterio: Basado en proximidad temporal de referencias: página residente menos recientemente usada como página a eliminar
- Implementación:Posible implementación con HW específico y un contador de accesos a memoria:
  - Cada entrada de la TP posee un contador
  - Cada acceso a memoria la MMU copia el contador del sistema a entrada referenciada
  - Reemplazo: página con contador más bajo
  - Difícil implementación estricta (hay aproximaciones): precisaría una MMU específica, ya que habría que controlar los accesos realizados a cada marco para actualizar los contadores de los accesos en la TP
- Nota: en su versión global, hay que considerar los contadores de las páginas menos recientemente usadas teniendo en cuenta el tiempo lógico de cada proceso

70 E) Buffering de páginas

- Criterio:Esta técnica intenta **evitar el problema con las páginas modificadas** que han de ser desalojadas. En este caso, el tratamiento del fallo de página implica realizar dos accesos a disco, uno para almacenar la página modificada y para traer la nueva

- Implementación:
  - Mantiene una **reserva de marcos libres**. Cuando se produce un fallo de página, siempre se usa un marco libre (es decir, en verdad no hay reemplazo)
  - Cuando el número de marcos libres queda por debajo de cierto **umbral** se activa un "demonio de paginación", que aplica repetidamente el algoritmo de reemplazo:
    - Páginas no modificadas pasan a ***lista de marcos libres***
    - Páginas modificadas pasan a ***lista de marcos modificados:*** cuando se escriban a disco pasan a lista de libres; suelen escribirse en tandas (lo que mejora el rendimiento)
  - Si se referencia una página mientras está en estas listas: se recupera directamente de la lista (no hay E/S), lo que puede mejorar el comportamiento de algoritmos poco eficientes

71 F) Retención de páginas en memoria

- Criterio:**No** todas las páginas son **reemplazables**
- Aplicación:
  - Se aplica a páginas del propio S.O: si sus páginas están fijas en memoria, su gestión es más sencilla
  - También se aplica mientras se hace DMA sobre una página. La página no será reemplazable hasta que finalice la operación sobre ella

Autoría del material

(c) Mario Muñoz Organero