Tema 3. Organización y Gestión de Memoria

1. **Conceptos generales**
2. **Organización de la Memoria Virtual**
3. **Gestión de la Memoria Virtual**

Bibliografía:

"Fundamentos de sistemas operativos : teoría y ejercicios resueltos", Candela Solá, S. [et al.], International Thomson Editores 2007.

"Sistemas Operativos.Aspectos Internos y Principios de Diseño", Prentice Hall, 2005. (Ver en concreto Apartados 8.1, 8.2)

En los puntos 1, 2 y 3 se estudia la materia a partir del punto en que se dejó en la asignatura de Fundamentos del Software; como preparación al tema repase lo que se estudió en Fundamentos del Software sobre este tema y resuelva todas las dudas que aparezcan.

Objetivos

- Conocer las distintas formas en las que el sistema operativo puede organizar y gestionar la memoria física
- Saber en qué consiste y para qué se utiliza el mecanismo de intercambio (*swapping*)
- Entender qué es la Memoria Virtual y por qué se utiliza
- Conocer los mecanismos de paginación, segmentación y segmentación paginada y cómo se llevan a cabo en un sistema con memoria virtual
- Comprender qué es la propiedad de localidad y su relación con el comportamiento de un programa en ejecución
- Conocer la teoría del conjunto de trabajo y el problema de la hiperpaginación
- Conocer aspectos de implementación en Linux de los conceptos anterioes

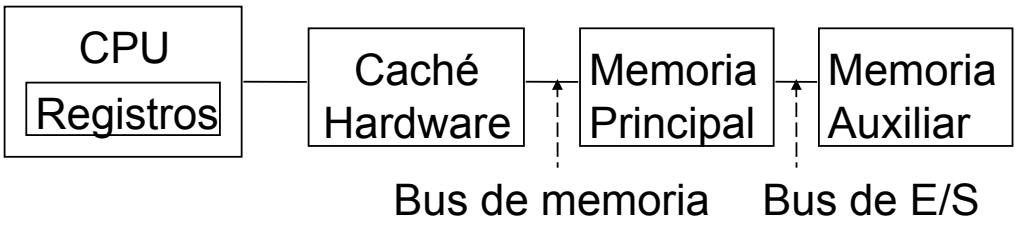
1.CONCEPTOS GENERALES
Jerarquía de Memoria

Dos principios sobre memoria:

① Menor cantidad, acceso más rápido, mayor coste por byte

② Mayor cantidad, acceso más lento, menor coste por byte

Así, los elementos frecuentemente accedidos se ponen en memoria rápida, cara y pequeña; el resto, en memoria lenta, grande y barata.



Conceptos sobre Cachés

- Caché - copia que puede ser accedida más rápidamente que el original
- Idea: hacer los casos frecuentes eficientes, los caminos infrecuentes no importan tanto
- *Acierto de caché*: item en la caché
- *Fallo de caché*: item no en caché; hay que realizar la operación completa
- **Tiempo de Acceso Efectivo** (TAE) =

  Probabilidad_acierto * coste_acierto +
  Probabilidad_fallo * coste_fallo
- Funciona porque los programas no son aleatorios, explotan la **localidad** → **principio de localidad**

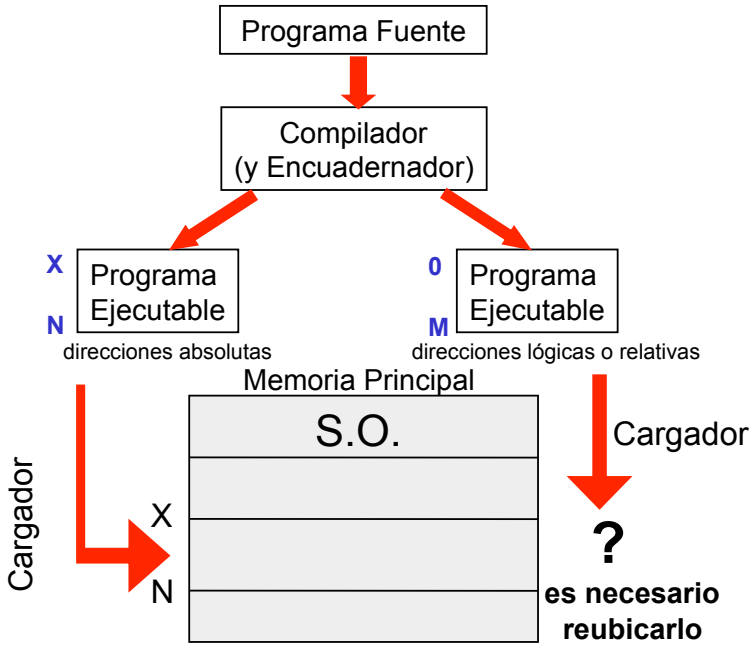
Objetivos a conseguir

- **Organización**: ¿cómo está dividida la memoria?
- **Gestión**: Dado un esquema de organización, ¿qué estrategias se deben seguir para obtener un rendimiento óptimo?
  - » Estrategias de **asignación**
  - » Estrategias de **sustitución (determinación de qué zona de memoria elegir para ser sustituida por otra)**
  - » Estrategias de **búsqueda**
- **Protección**
  - » El SO de los procesos de usuario
  - » Los procesos de usuario entre ellos

Carga absoluta y Reubicación

Carga absoluta
Se asignan direcciones físicas en tiempo de compilación

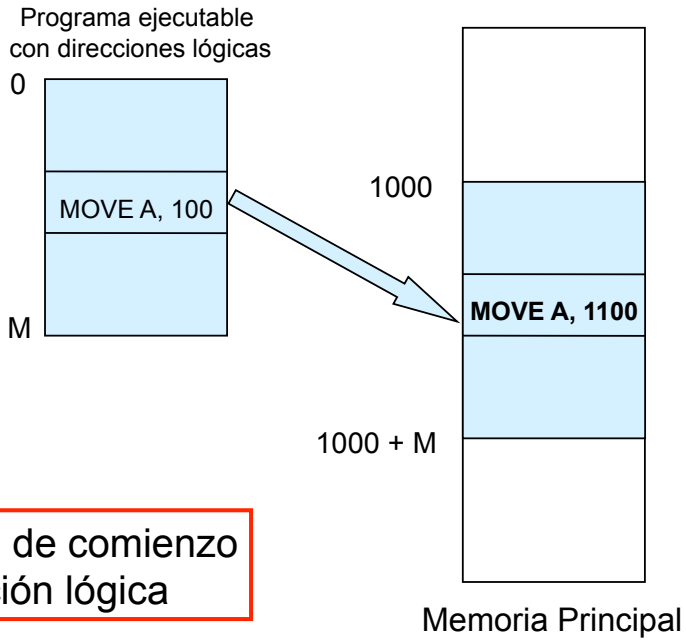
Reubicación
Capacidad de cargar y ejecutar un programa en un lugar arbitrario de la memoria



Reubicación estática

• Decisión de dónde ubicar el programa en **tiempo de carga**

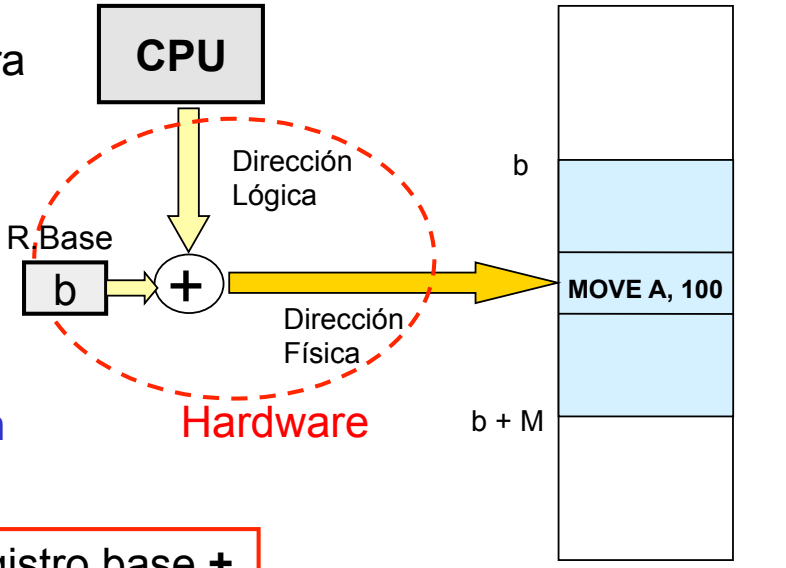
• El compilador genera **direcciones lógicas** (relativas)



dirección física = dirección de comienzo
+ dirección lógica

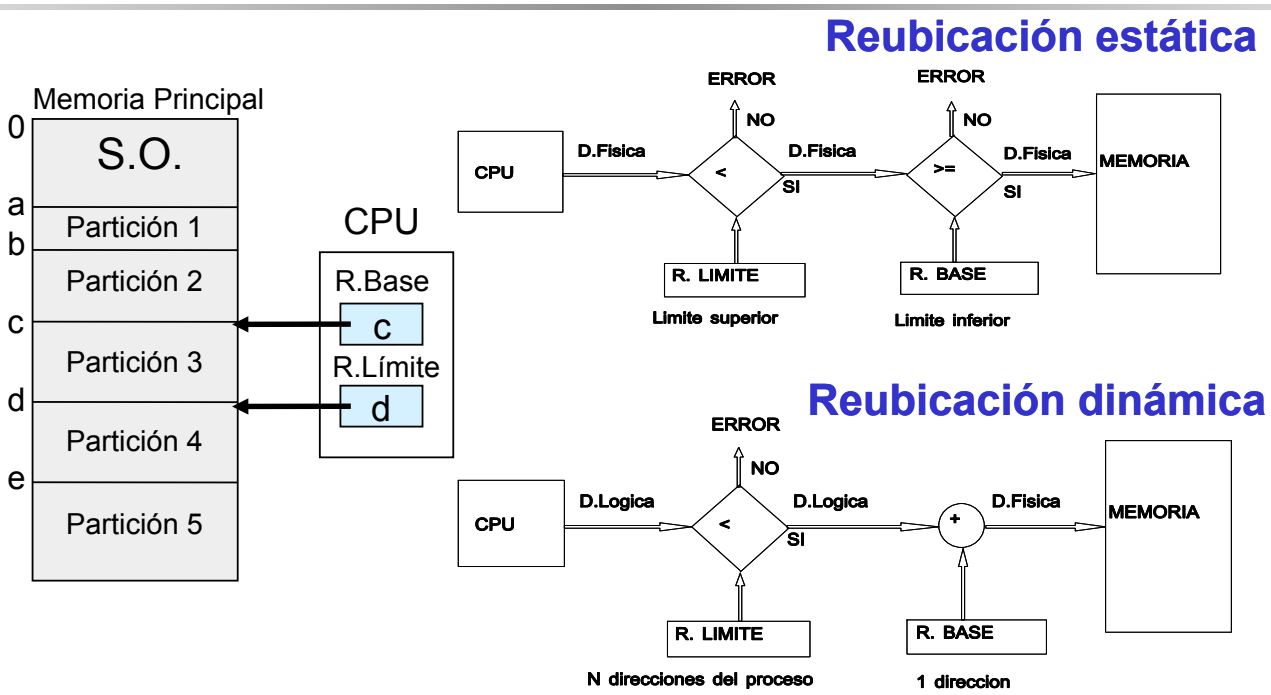
Concepto: Reubicación dinámica

- El compilador genera **direcciones lógicas** (relativas)
- La traducción de direcciones lógicas a físicas se hace en **tiempo de ejecución**



dirección física = registro base + dirección lógica

Protección en asignación contigua



Intercambio (*Swapping*)

- Intercambiar procesos entre memoria y un almacenamiento auxiliar
- El almacenamiento auxiliar debe ser un disco rápido con espacio para albergar las imágenes de memoria de los procesos de usuario
- El factor principal en el tiempo de intercambio es el tiempo de transferencia
- El ***intercambiador*** tiene las siguientes responsabilidades:
  - » Seleccionar procesos para retirarlos de MP
  - » Seleccionar procesos para incorporarlos a MP
  - » Gestionar y asignar el espacio de intercambio

Localización del espacio de intercambio

- Se mantiene un archivo de intercambio global con la información de intercambio de todos los procesos
- Existe un archivo de intercambio para cada proceso

Problema: Necesita más espacio en disco, los accesos son más lentos

2. ORGANIZACION DE LA MEMORIA VIRTUAL

① Concepto de memoria virtual

② Paginación

③ Segmentación

④ Segmentación paginada

2.1 Concepto de memoria virtual

- Memoria Virtual
  - » El tamaño del programa, los datos y la pila puede exceder la cantidad de memoria física disponible para él.
  - » Se usa un almacenamiento a dos niveles:
    - – *Memoria Principal* ➜ partes del proceso necesarias en un momento dado
    - – *Memoria Secundaria* ➜ espacio de direcciones completo del proceso

Concepto de memoria virtual (y II)

» Es necesario:

– saber qué se encuentra en memoria principal

– una política de movimiento entre MP y MS

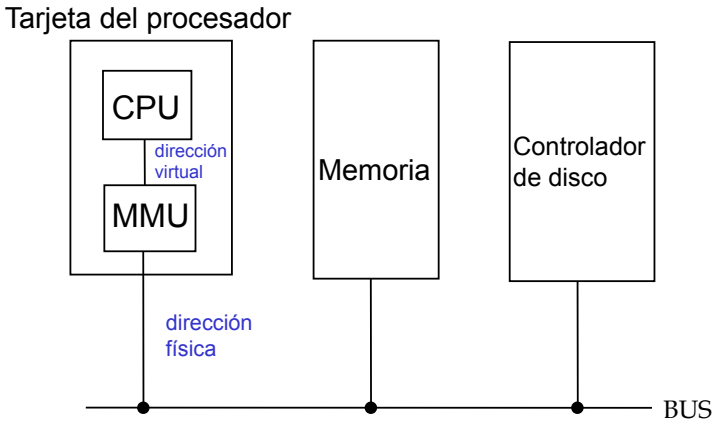
- Además, la memoria virtual
  - » resuelve el problema del crecimiento dinámico de los procesos
  - » puede aumentar el grado de multiprogramación

Unidad de Gestión de Memoria

- La **MMU** (*Memory Management Unit*) es un dispositivo hardware que traduce direcciones virtuales a direcciones físicas ¡Este dispositivo está gestionado por el SO!
- En el esquema MMU más simple, el valor del registro base se añade a cada dirección generada por el proceso de usuario al mismo tiempo que es enviado a memoria
- El programa de usuario trata sólo con direcciones lógicas; éste nunca ve direcciones reales

Unidad de Gestión de Memoria (y II)

- Además de la traducción, el MMU deberá:
  - detectar si la dirección aludida se encuentra o no en MP
  - generar una excepción si no se encuentra en MP

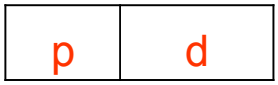


2.2 Paginación

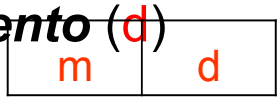
- El espacio de direcciones físicas de un proceso puede ser no contiguo
- La memoria física se divide en bloques de tamaño fijo, denominados *marcos de página*. El tamaño es potencia de dos, de 0.5 a 8 Kb
- El espacio lógico de un proceso se divide en bloques del mismo tamaño, denominados *páginas*
- Los marcos de páginas contendrán páginas de los procesos

Paginación (y II)

- Las **direcciones lógicas**, que son las que genera la CPU se dividen en ***número de página*** (p) y ***desplazamiento*** dentro de la página (d)



- Las **direcciones físicas** se dividen en ***número de marco*** (m, *dirección base del* marco donde está almacenada la página) y ***desplazamiento*** (d)



Paginación (y III)

- Cuando la CPU genere una dirección lógica será necesario traducirla a la dirección física correspondiente, la *tabla de páginas* mantiene información necesaria para realizar dicha traducción. *Existe una tabla de páginas por proceso*
- *Tabla de ubicación en disco* (una por proceso) ubicación de cada página en el almacenamiento secundario
- *Tabla de marcos de página,* usada por el S.O. y contiene información sobre cada marco de página

Contenido de la tabla de páginas

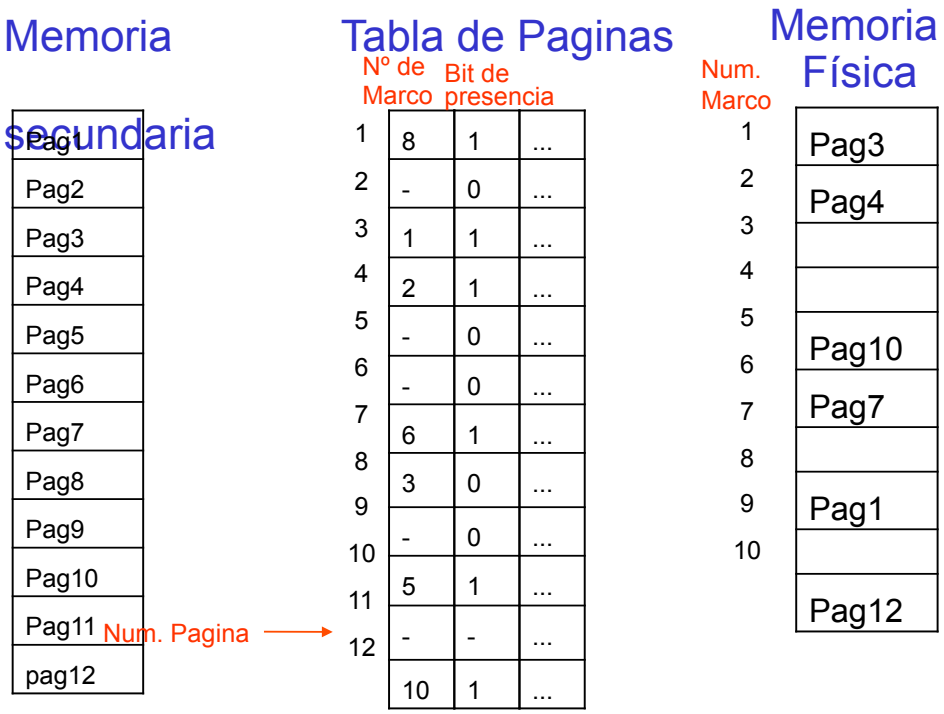
Una entrada por cada página del proceso:

- Número de marco (dirección base del marco) en el que está almacenada la página si está en MP
- Bit de presencia o bit válido
- Bit de modificación
- Modo de acceso autorizado a la página (bits de protección)

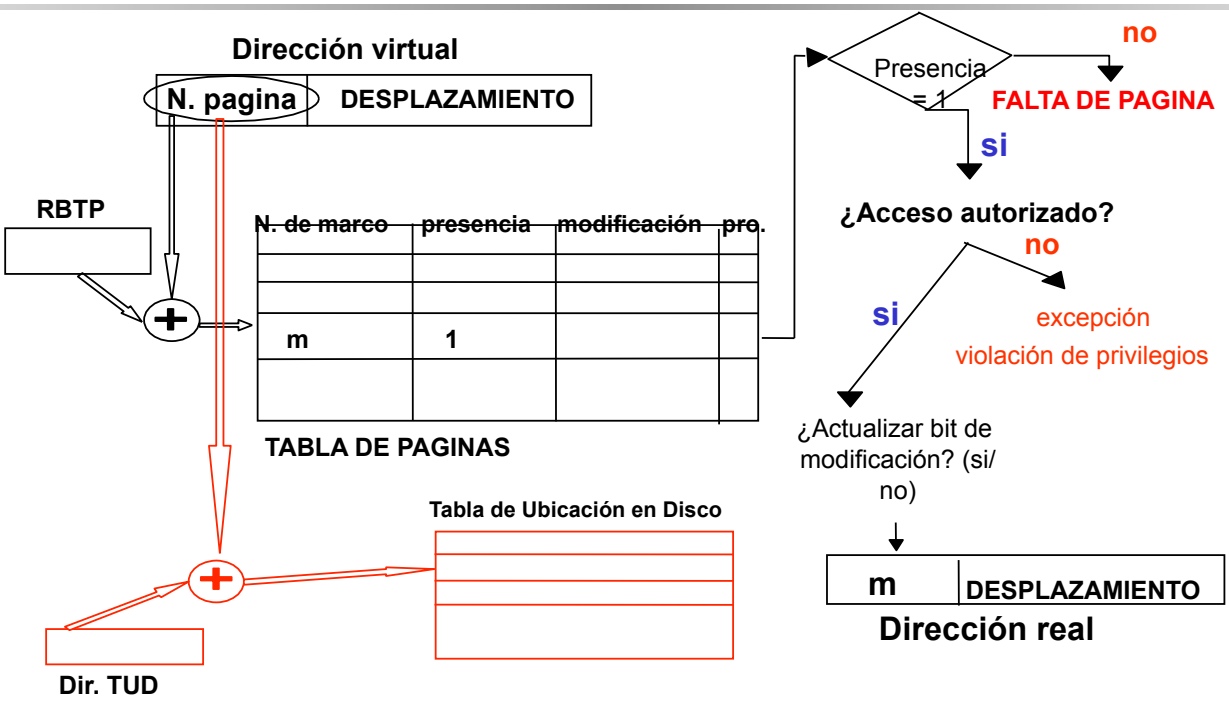
Nº de página

nº marco	presencia	modificación	protección
46	1	0	01

Ejemplo: contenido de la tabla de páginas



Esquema de traducción



Falta de página

1. Bloquear proceso
2. Encontrar la ubicación en disco de la página solicitada (*tabla de ubicación en disco*)
3. Encontrar un marco libre. Si no hubiera, se puede optar por desplazar una página de MP
4. Cargar la página desde disco al marco de MP
5. Actualizar tablas (bit presencia=1, nº marco, ...)
6. Desbloquear proceso
7. Reiniciar la instrucción que originó la falta de página

Implementación de la Tabla de Páginas

- La tabla de páginas se mantiene en memoria principal
- El *registro base de la tabla de páginas* (*RBTP*) apunta a la tabla de páginas (suele almacenarse en el PCB del proceso)
- En este este esquema:
  - » cada acceso a una instrucción o dato requiere dos accesos a memoria: uno a la tabla de páginas y otro a memoria
  - » un problema adicional viene determinado por el tamaño de la tabla de páginas

Tamaño de la Tabla de Páginas

- Ejemplo:
  - » Dirección virtual: 32 bits.
  - » Tamaño de página = 4 Kbytes (2^{12} bytes).

⇨ tamaño del campo desplazamiento = **12 bits**

⇨ tamaño número de página virtual = **20 bits**

⇨ Nº de páginas virtuales = 2^{20} = **¡1,048,576!**

- Solución para reducir el tamaño de la TP:
  - » Paginación multinivel

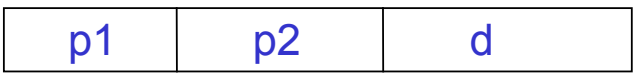
Paginación multinivel

- Esta solución opta por “paginar las tablas de páginas”

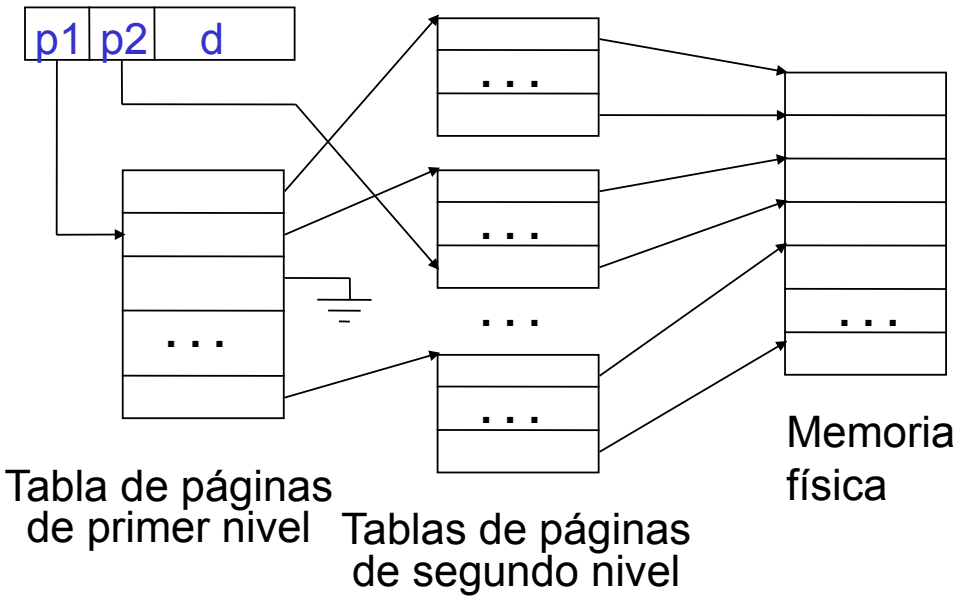
- La partición de la tabla de páginas permite al SO dejar particiones no usadas sin cargar hasta que el proceso las necesita. Aquellas porciones del espacio de direcciones que no se usan no necesitan tener tabla de página

Paginación a dos niveles

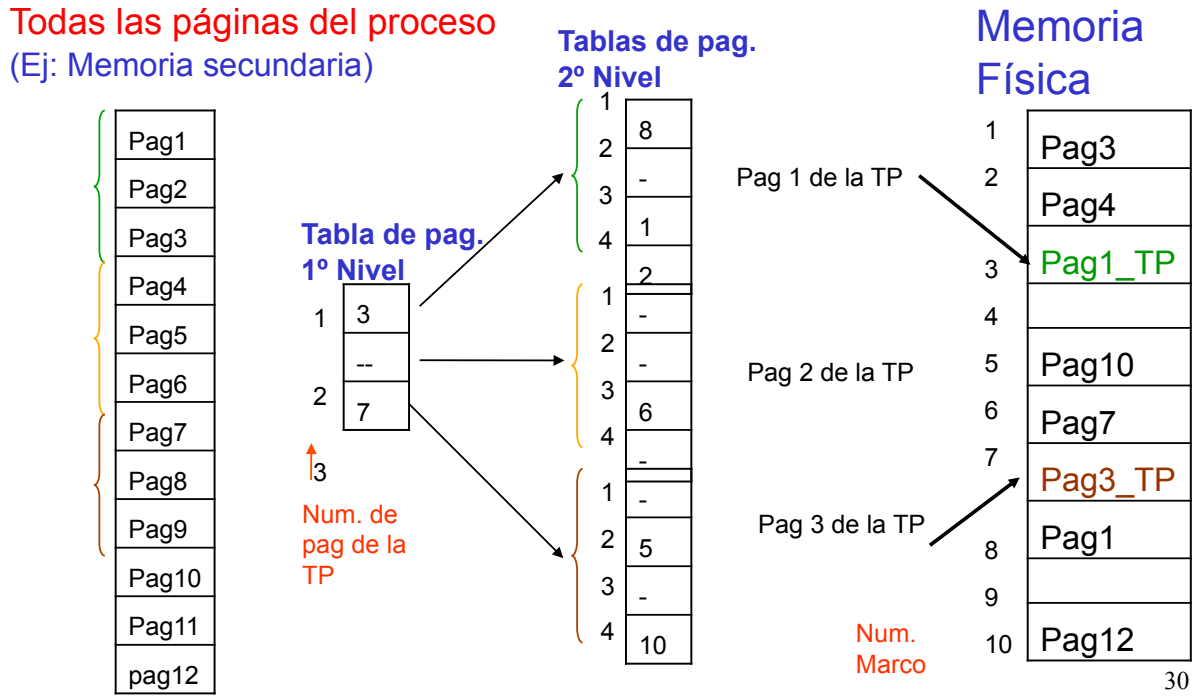
- Lo que hacemos es paginar la tabla de páginas.
- La dirección lógica se divide en:
  - » número de página (n bits):
    - – un número de página p1 (=k)
    - – desplazamiento de página p2 (=n-k)
  - » desplazamiento de página d (m bits)
- Así una dirección lógica es de la forma:



Esquema de paginación a dos niveles

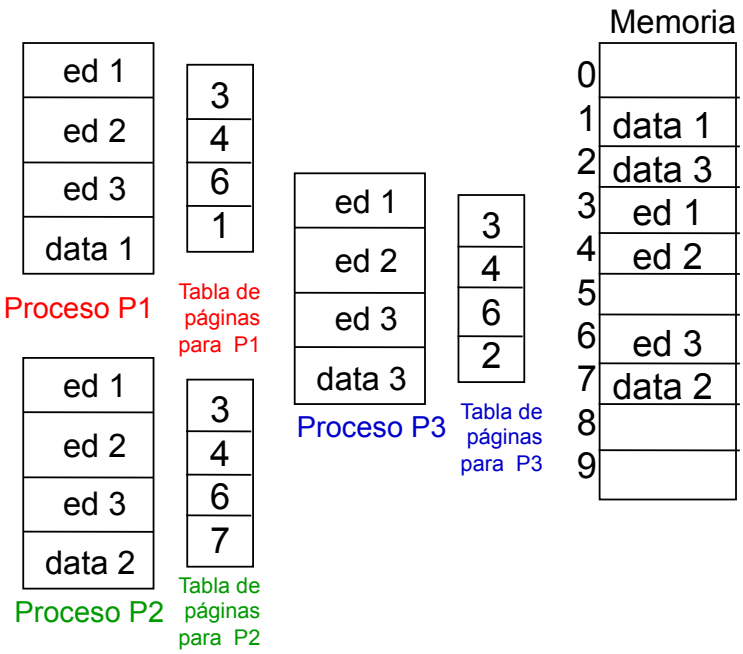


Ejemplo: Esquema de paginación a dos niveles



Páginas compartidas

- Una copia de código de solo lectura compartido entre varios procesos.
  Ej. editores, compiladores, sistemas de ventanas



2.3 Segmentación

- Esquema de organización de memoria que soporta mejor la visión de memoria del usuario: un programa es una colección de unidades lógicas -segmentos-, p. ej. procedimientos, funciones, pila, tabla de símbolos, matrices, etc.

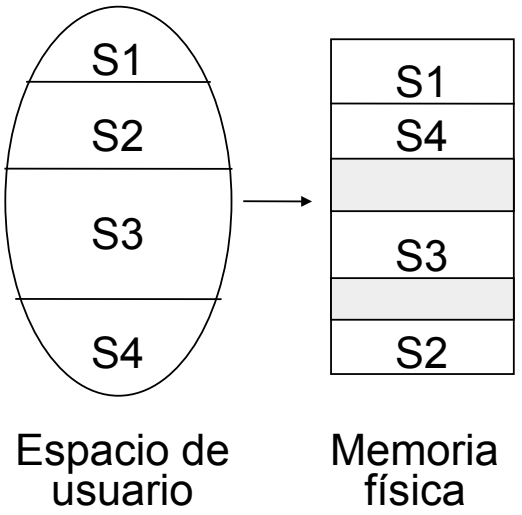


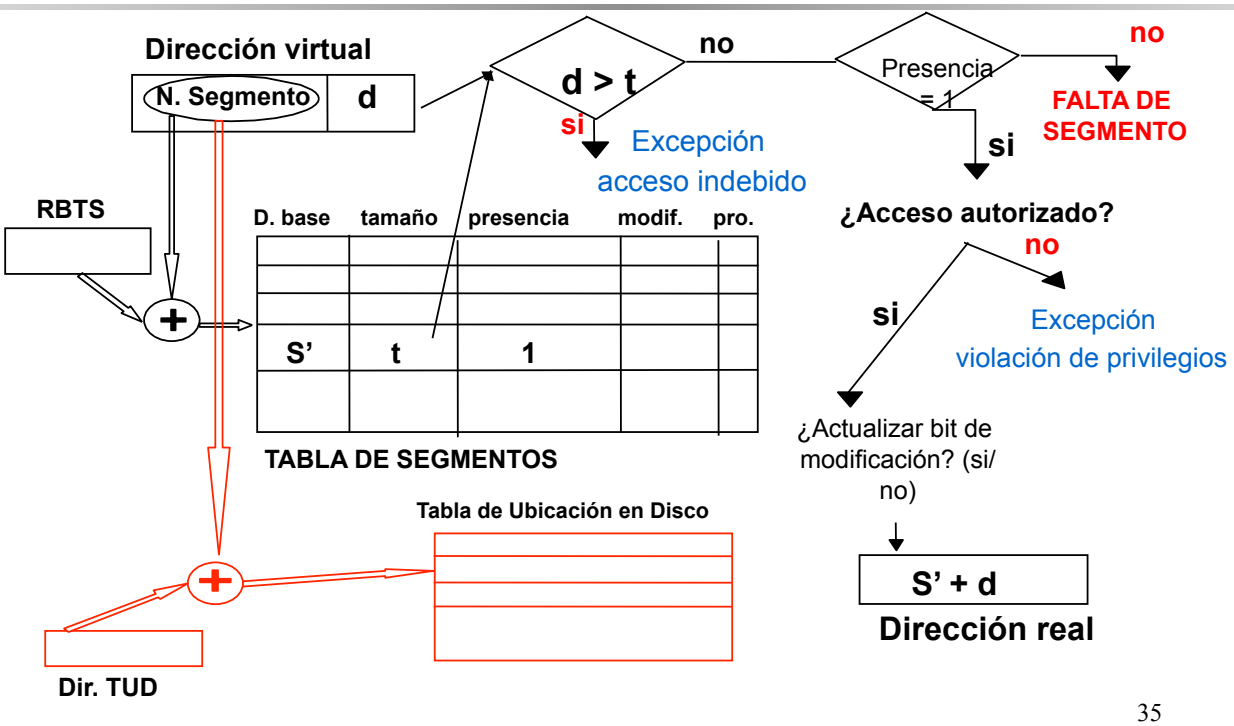
Tabla de Segmentos

- Una dirección lógica es una tupla:
  <número_de_segmento, desplazamiento>
- La *Tabla de Segmentos* aplica direcciones bidimensionales definidas por el usuario en direcciones físicas de una dimensión. Cada entrada de la tabla tiene los siguientes elementos (aparte de presencia, modificación y protección):
  - » *base* - dirección física donde reside el inicio del segmento en memoria.
  - » *tamaño* - longitud del segmento.

Implementación de la Tabla de Segmentos

- La tabla de segmentos se mantiene en memoria principal
- El *Registro Base de la Tabla de Segmentos* (*RBTS*) apunta a la tabla de segmentos (suele almacenarse en el PCB del proceso)
- El *Registro Longitud de la Tabla de Segmentos* (*STLR*) indica el número de segmentos del proceso; el nº de segmento *s,* generado en una dirección lógica, es legal si $s < STLR$ (suele almacenarse en el PCB del proceso)

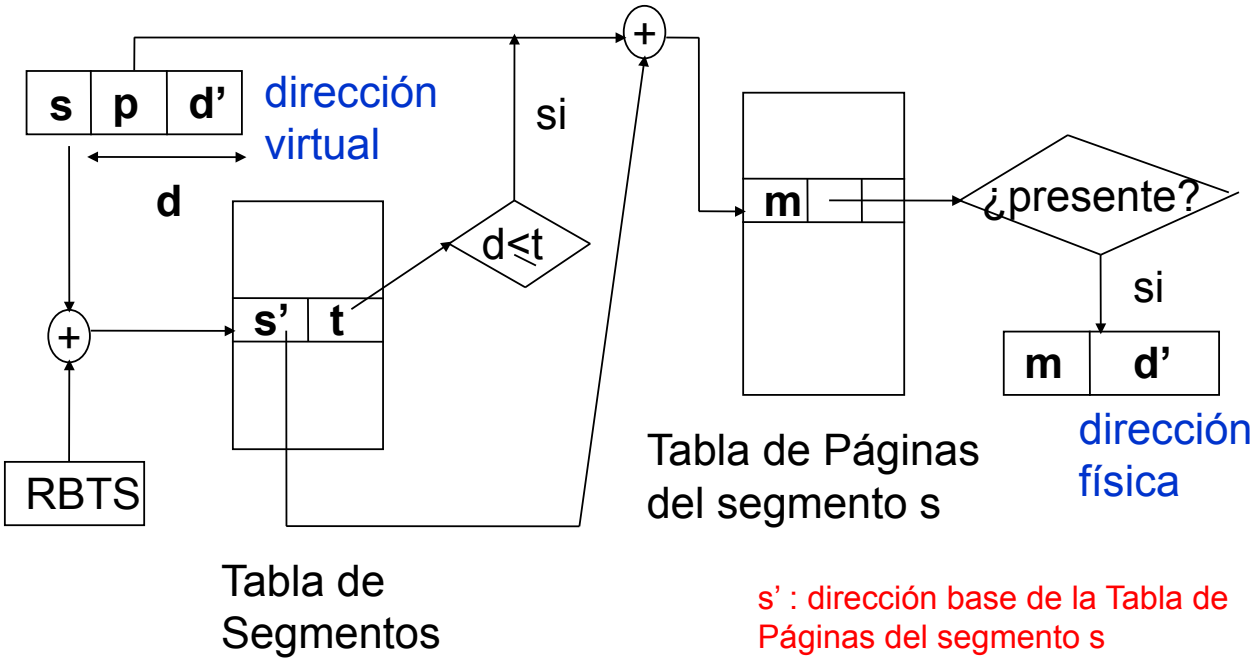
Esquema de traducción



2.4 Segmentación Paginada (No entran problemas)

- La variabilidad del tamaño de los segmentos y el requisito de memoria contigua dentro de un segmento complica la gestión de MP y MS
- Por otro lado, la paginación simplifica eso pero complica más los temas de compartición y protección (éstos van mejor en segmentación) .
- Algunos sistemas combinan ambos enfoques, obteniendo la mayoría de las ventajas de la segmentación y eliminando los problemas de una gestión de memoria compleja

Esquema de traducción



s' : dirección base de la Tabla de Páginas del segmento s

3. Gestión de la Memoria Virtual

① Introducción

② Algoritmos de sustitución

③ Comportamiento de los programas

④ Hiperpaginación (*Thrashing*)

⑤ Algoritmos de asignación y sustitución

3.1 Introducción

- Gestión de Memoria Virtual con paginación
- Criterios de clasificación respecto a:
  - » *Políticas de asignación*:
    - –Fija
    - –Variable
  - » *Políticas de búsqueda*:
    - –Paginación por demanda
    - –Paginación anticipada
  - » *Políticas de sustitución*:
    - –Sustitución local
    - –Sustitución global

Introducción (y II)

- Independientemente de la **política de sustitución** utilizada, existen ciertos criterios que siempre deben cumplirse:
  - » Páginas “limpias” frente a “sucias”
    - –se pretende minimizar el coste de transferencia
  - » Páginas compartidas
    - –se pretende reducir el nº de faltas de página
  - » Páginas especiales
    - –algunos marcos pueden estar bloqueados (ejemplo: buferes de E/S mientras se realiza una transferencia)

Asignación fija

- Asignación por igual
  - » Se asignan el mismo número de marcos a todos los procesos
  - » Si hay m marcos, y n procesos. A cada proceso se se asignan *m/n* marcos
- Asignación proporcional por tamaño
  - » Asigna según tamaño del proceso
  - » s_i=tamaño de p_i
  - » $S = \Sigma s_i$
  - » m = número total de marcos
  - » la asignación, a_i, para p_i es:
    $a_i = (s_i / S)*m$

Paginación por demanda frente a anticipada

- Las ventajas de la paginación **por demanda** son:
  - » Se garantiza que en MP solo están las páginas necesarias en cada momento
  - » La sobrecarga de decidir qué páginas llevar a MP es mínima
- Las ventajas de la paginación **anticipada** son:
  - » Se puede optimizar el tiempo de respuesta para un proceso pero los algoritmos son más complejos y se consumen más recursos

Rendimiento de la paginación por demanda

- Sea *p* la tasa de falta de página; p=0 no hay faltas de páginas, ó p=1, toda referencia es una falta. Por tanto, $0 \leq p \leq 1$
- **TAE** = (1 - p) * acceso_a_memoria
  + p * (sobrecarga_falta_de_página
  + [sacar_fuera_una_página]
  + traer_la_página
  + sobrecarga_de_rearranque)

Influencia del tamaño de página

- Cuanto más pequeñas
  - » Aumento del tamaño de las tablas de páginas
  - » Aumento del nº de transferencia MP↔Disco
  - » Reducen la fragmentación interna
- Cuanto más grandes
  - » Grandes cantidades de información que no serán usadas están ocupando MP
  - » Aumenta la fragmentación interna
- Búsqueda de un equilibrio

3.2 Algoritmos de sustitución

- Podemos tener las siguientes combinaciones
  - » asignación fija y sustitución local
  - » asignación variable y sustitución local
  - » asignación variable y sustitución global
- Veremos distintos algoritmos de sustitución y nos basaremos (por simplicidad) en que se utiliza una política de asignación fija y sustitución local
- *Cadena de referencia,* $\omega = r_1, r_2, r_3, \ldots, r_i, \ldots,$: secuencia de números de páginas referenciadas por un proceso durante su ejecución

A) Algoritmo Optimo

- Se sustituye la página que no será objeto de ninguna referencia posterior o que se referencie más tarde

» 4 marcos de página
» Problema: debemos tener un "conocimiento perfecto" de la cadena de referencia
» Se utiliza para medir cómo de bien se comportan otros algoritmos
» Faltas de páginas: 6

1,2,3,4,1,2,5,1,2,3,4,5

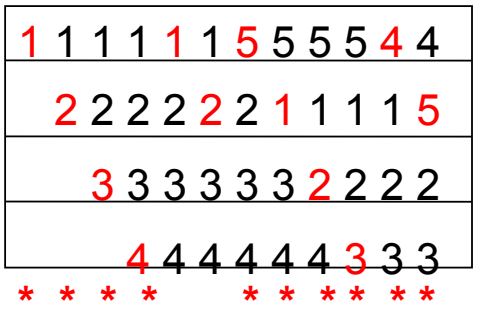
1	1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5	5
*	*	*	*			*				*	

B) Algoritmo FIFO

- Se sustituye la página por orden cronológico de llegada a MP (la página más antigua)

» 4 marcos de página
» Faltas de página: 10
» Sufre de la *Anomalía de Belady*: "más marcos no implican menos faltas de páginas"

1,2,3,4,1,2,5,1,2,3,4,5

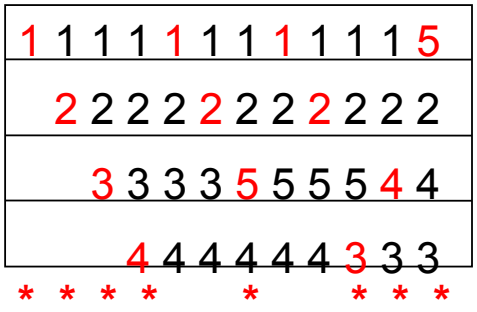


C) Algoritmo LRU

- Se sustituye la página que fue objeto de la referencia más antigua (*Least Recently Used*)

» 4 marcos de página
» Faltas de página: 8
» Implementación del algoritmo:
  – con contadores
  – con pila
» Mayor coste

1,2,3,4,1,2,5,1,2,3,4,5

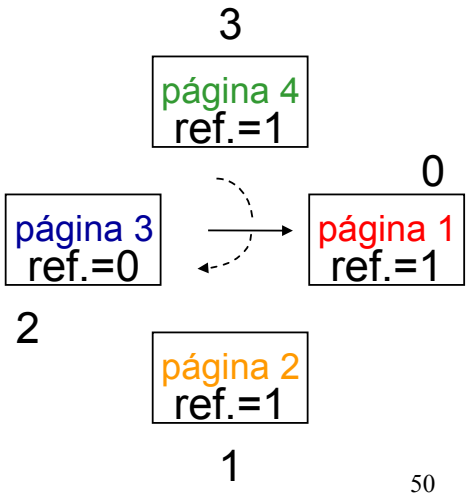


Implementaciones de LRU

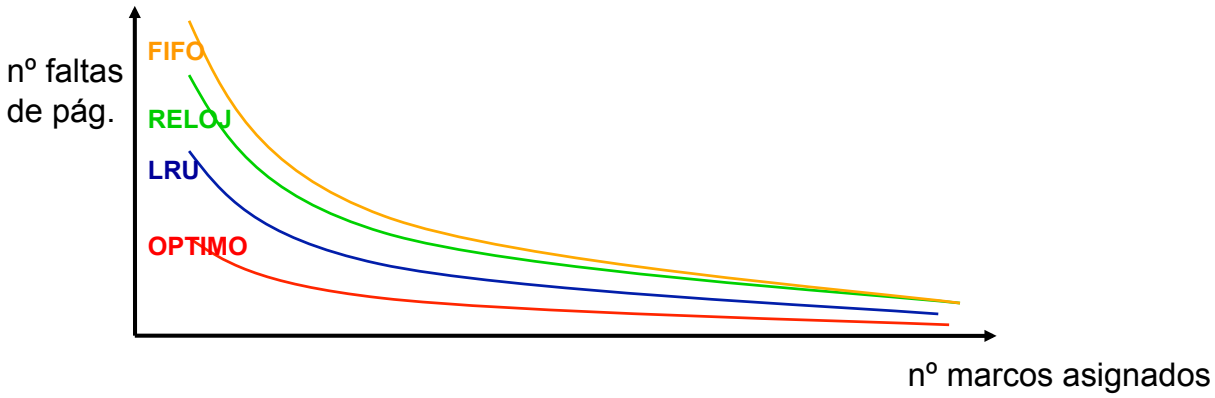
- LRU con contador
  - » Cada entrada de la tabla de páginas tiene un contador. Cada vez que se referencia la página, se copia el tiempo del reloj en el contador
  - » Cuando necesitamos cambiar una página, se miran todos los contadores y se elige la que tiene el menor tiempo
- LRU con pila
  - » Los números de páginas se mantienen en una pila (lista doblemente enlazada). Cuando se referencia una página se mueve a la cima de la pila (cambio de seis punteros como máximo)
  - » No hay que hacer búsqueda para la sustitución de una página, se sustituye la del fondo de la pila

D) Algoritmo del reloj

- Cada página tiene asociado un bit de referencia R
- Inicialmente R= 0; en cada referencia el hardware lo pone a 1
- Los marcos de página se representan por una lista circular y un puntero a la página visitada hace más tiempo
- Selección de una página:
  1. Consultar marco actual
  2. ¿Es R=1?
     - –Si: R=0; ir al siguiente marco y volver al paso 1
     - –No: seleccionar para sustituir e incrementar posición



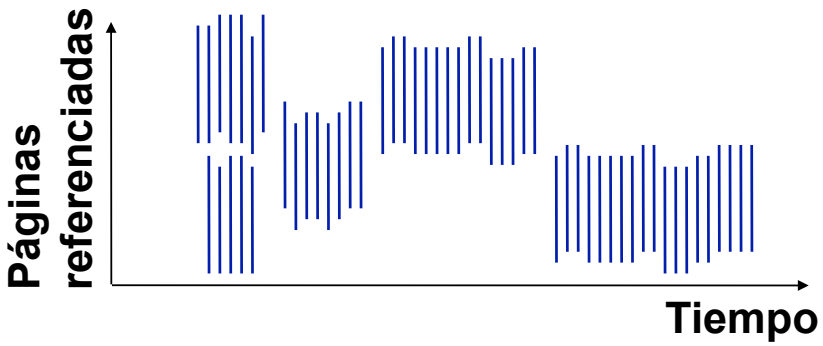
Comparación



- Conclusión:
  - » Influye más la cantidad de MP disponible que el algoritmo de sustitución usado

3.3 Comportamiento de los programas

- Viene definido por la secuencia de referencias a página que realiza el proceso
- Importante para maximizar el rendimiento del sistema de memoria virtual (TLB, alg. sustitución, ...)



Propiedad de localidad

- Distintos tipos
  - » Temporal: Una posición de memoria referenciada recientemente tiene una probabilidad alta de ser referenciada en un futuro próximo (ciclos, rutinas, variables globales, ...)
  - » Espacial: Si cierta posición de memoria ha sido referenciada es altamente probable que las adyacentes también lo sean (array, ejecución secuencial, ...)

Conjunto de Trabajo

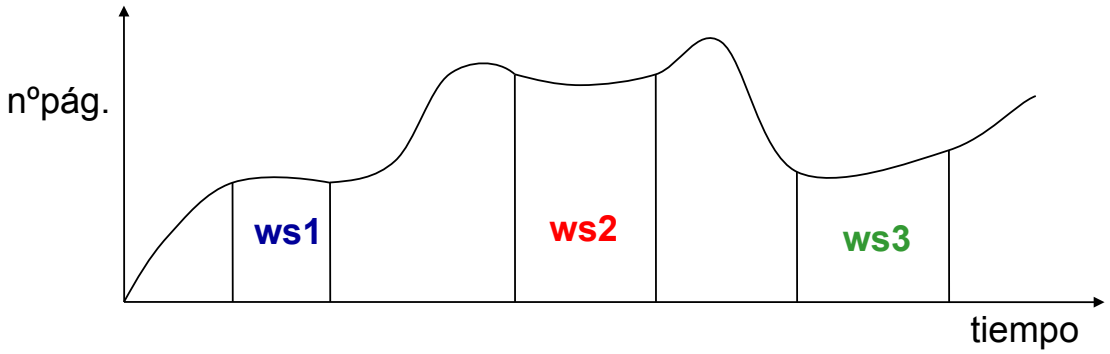
- Observaciones:
  - » Mientras el conjunto de páginas necesarias puedan residir en MP, el nº de faltas de página no crece mucho
  - » Si eliminamos de MP páginas de ese conjunto, la activación de paginación crece mucho
- Conjunto de trabajo *(Working Set)* de un proceso es el conjunto de páginas que son referenciadas frecuentemente en un determinado intervalo de tiempo

  Se define el conjunto de trabajo en el momento t con "ancho de ventana" Δ como:

  WS(t,Δ) = páginas referenciadas en el intervalo de tiempo **(t - Δ, t]**

Conjunto de Trabajo: propiedades

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

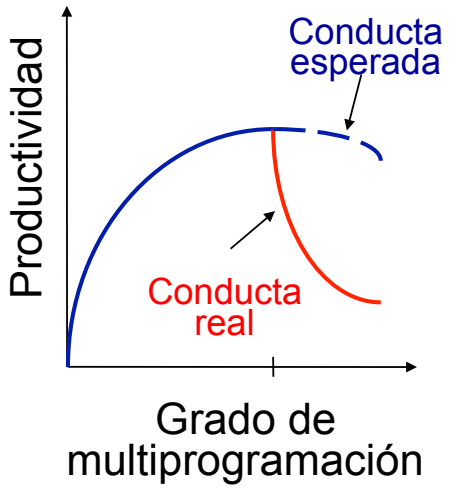


Teoría del Conjunto de Trabajo

- Un proceso sólo puede ejecutarse si su conjunto de trabajo está en memoria principal
- Una página no puede retirarse de memoria principal si está dentro del conjunto de trabajo del proceso en ejecución

3.4 Hiperpaginación

- Si un proceso no tiene "suficientes" páginas, la tasa de faltas es alta
  - » bajo uso de la CPU
  - » el SO incrementa el grado de multiprogramación
  - » más faltas de páginas
- *Hiperpaginación* = el sistema operativo está ocupado en resolver faltas de página



Hiperpaginación (y II)

- Formas de evitar la hiperpaginación:
  - » Asegurar que cada proceso existente tenga asignado un espacio en relación a su comportamiento ➔ Algoritmos de asignación variables
  - » Actuar directamente sobre el grado de multiprogramación ➔ Algoritmos de regulación de carga

3.5 Algoritmos de asignación y sustitución variables:
A) Algoritmo basado en el modelo del WS

Parámetro del algoritmo: ancho de ventana V

En cada momento t en que se hace referencia a una posición de memoria, se determina el conjunto de trabajo de la siguiente forma:

WS = páginas referenciadas en el intervalo **(t - V, t]**

solo estas páginas se mantienen en memoria principal

Ejemplo de algoritmo basado en el WS con V = 4

En t=-2 se referencia E
En t=-1 se referencia D
En t_0 se referencia A, y WS={A,D,E}
Comenzamos a ejecutar el algoritmo en t=1

(t-V, t]				(-3,1]	(-2,2]	(-1,3]	(0,4]	(-1,5]	(2,6]	(3,7]	(4,8]	(5,9]	(6,10]
WS=			ADE	EDAC	DAC	ACD	CDB	DBCE	DBCE	BCE	CE	ECA	CEAD
Valores de t:	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Secuencia de págs													
que se referencian:---->----->				**C**	**C**	**D**	**B**	**C**	**E**	**C**	**E**	**A**	**D**
A			A	A	A	A						A	A
B							B	B	B	B			
C				C	C	C	C	C	C	C	C	C	C
D		D	D	D	D	D	D	D	D				D
E	E		E	E					E	E	E	E	E
* = Hay FP				*			*		*			*	*

B) Algoritmo FFP
(Frecuencia de Falta de Página)

Definición: Intervalo entre dos faltas de página =
Tiempo actual – tiempo de la falta de página anterior

IF intervalo > valor umbral L // L es un parámetro del algoritmo
THEN
//el proceso tiene demasiados marcos de página, hay que quitarle.

se retiran de memoria principal todas las páginas no referenciadas entre el momento actual y el momento de la falta de página anterior

ELSE // demasiadas faltas de página, el proceso tiene pocos marcos.
se asigna al proceso un marco de página más para albergar la página que ha originado esta falta de página

LA VELOCIDAD DE FP SIRVE PARA CONOCER SI EL Nº DE MARCOS ASIGNADOS AL PROCESO ES GRANDE O PEQUEÑO

Ejemplo de algoritmo FFP con valor L = 2

En t=-2 se referencia E
En t= -1 se referencia A
En t=0 se referencia D
Comenzamos a ejecutar el algoritmo en t=1

				Se añade C					Se añade E					Se añade D
¿ Intervalo entre... > L?				NO			SI		NO				SI	NO
Intervalo de tentre														
esta esta FP														
y la anterior:				1			3		2					
Valores de tiempo:	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Páginas que se referencian:			**D**	**C**	**C**	**D**	**B**	**C**	**E**	**C**	**E**	**A**	**D**	
A		A	A	A	A	A						A	A	
B							B	B	B	B	B			
C				C	C	C	C	C	C	C	C	C	C	
D			D	D	D	D	D	D	D	D	D		D	
E	E		E	E	E	E			E	E	E	E	E	
* = Hay FP	*	*	*	*			*		*			*	*	