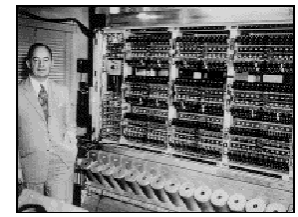
# Gestión de memoria

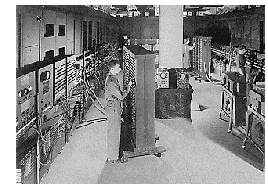
## Evolución sistemas operativos

### Sistemas interactivos o procesos en serie

(finales de la década de 1940)

- No existían los sistemas operativos
- Los programadores/operadores debían interactuar con el hardware del computador sin ayuda externa
- Accedían directamente a la consola de la computadora
- Se actuaba sobre una serie de micro interruptores que permitían introducir directamente el programa en la memoria de la computadora (en una dirección fija)

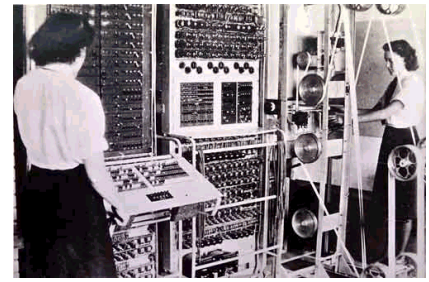




# Evolución sistemas operativos

## Sistemas interactivos o procesos en serie (cont.)

- Los programadores/ operadores
  - cargan un compilador,
  - un programa fuente,
  - salvan el programa compilado,
  - y cargan el ejecutable
- E/S mediante tarjetas perforadas
- Uso por turnos del ordenador



# Evolución sistemas operativos

## Sistemas de colas simples o procesos por lotes

(década de 1950)

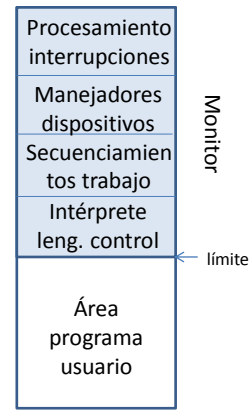
Antecesores SO:

- Monitor (residente en memoria)
- Procesamiento por lotes
- Almacenamiento temporal

# Evolución sistemas operativos

## Sistemas de colas simples o procesos por lotes (cont.)

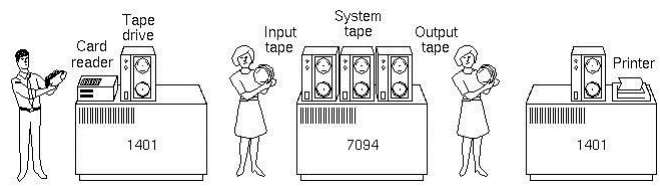
- **Monitor (residente en memoria)**
  - Software que automatiza las funciones del operador
  - Carga los programas a memoria, leyéndolos de una cinta o de tarjetas perforadas, y los ejecuta.
  - La memoria se divide en dos partes:
    - » Monitor
    - » Programa que se está ejecutando




# Evolución sistemas operativos

## Sistemas de colas simples o procesos por lotes (cont.)

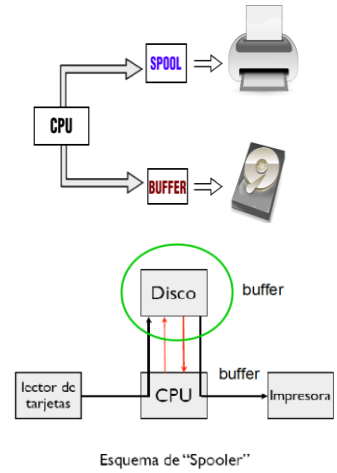
- **Procesamiento por lotes**
  - En una misma cinta o conjunto de tarjetas, se cargaban varios programas
  - Se ejecutaban uno a continuación de otro sin perder apenas tiempo en la transición
  - Con cada trabajo se incluye un conjunto de instrucciones para el monitor



# Evolución sistemas operativos

## Sistemas de colas simples o procesos por lotes (cont.)

- **Almacenamiento temporal**
  - Objetivo: disminuir el tiempo de carga de los programas
  - Simultaneaba la carga del programa o la salida de datos con la ejecución de la siguiente tarea.
  - Para ello se utilizaban dos técnicas, el buffering y el spooling.



Esquema de "Spooler"

# Evolución sistemas operativos

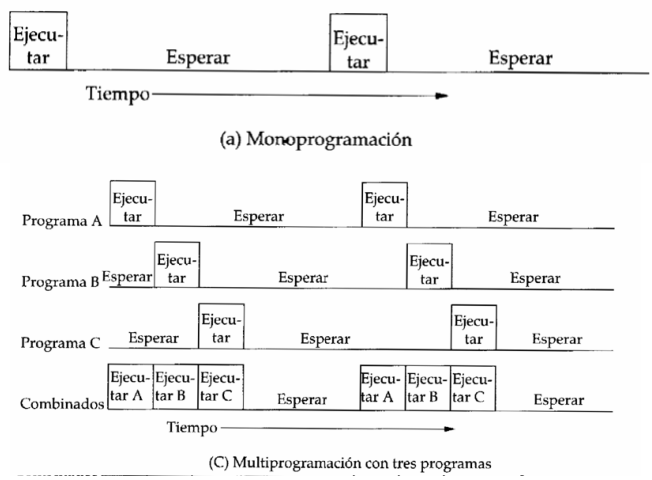
## Sistemas por lotes multiprogramados

(década de 1960)

- Aparecen los SO
- **Multiprogramación (o multitarea)**
  - La memoria principal alberga múltiples programas de usuario.
  - Al realizar una operación de E/S los programas ceden la CPU a otro programa
  - Algoritmos de planificación para que el procesador elija qué programa ejecuta

# Evolución sistemas operativos

## Sistemas por lotes multiprogramados (cont.)




# Evolución sistemas operativos

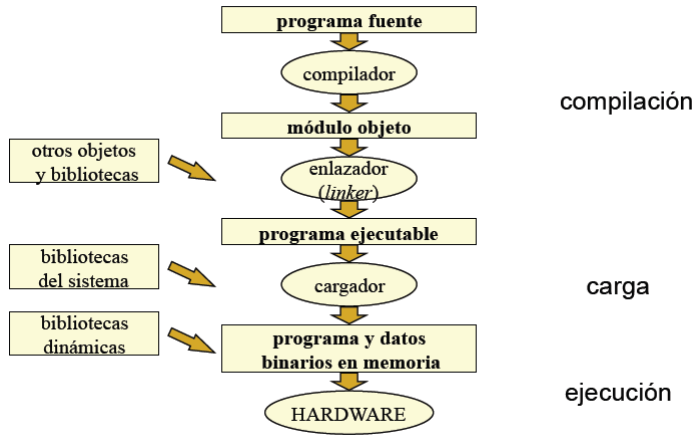
## Sistemas de tiempo compartido

- Se comparte el tiempo de procesador entre múltiples usuarios
- Cada usuario tiene una terminal on-line, donde puede enviar comandos y ver su salida en forma interactiva.
- El sistema operativo entrelaza la ejecución de cada programa de usuario en pequeños intervalos de tiempo o cuantos de computación (no sólo cuando se espera por una operación de E/S)
- Cada usuario tiene la apariencia de que posee la maquina para él solo.

# Tareas de la gestión de memoria

- Controlar qué partes de la memoria están utilizadas o libres.
- Asignar memoria a procesos y liberarla cuando terminan.
- Administrar intercambio entre memoria principal y disco (Memoria Virtual).

# Enlace de direcciones

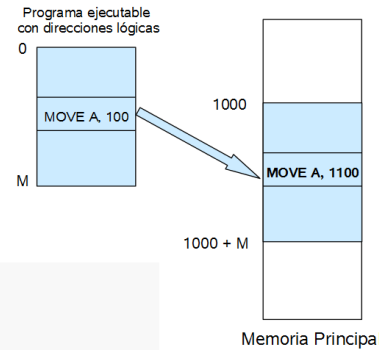


# Enlace de direcciones

1. **En tiempo de compilación:**
   - El compilador traduce direcciones de memoria *simbólicas* a direcciones binarias.
   - Si las direcciones binarias son absolutas, el programa sólo se puede ejecutar en una zona fija de la memoria
   - Ej.: los programas con formato .COM de MSDOS

# Enlace de direcciones

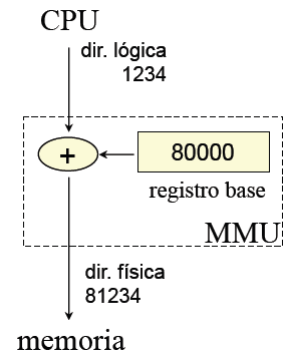
2. **En tiempo de carga** (Reubicación estática):
   - El compilador genera código reubicable.
   - Se crean direcciones de memoria absolutas cuando se carga el programa en memoria.
   - Lo realiza el monitor con un módulo llamado cargador *(loader)*




dirección física = dirección de comienzo
+ dirección lógica

# Enlace de direcciones

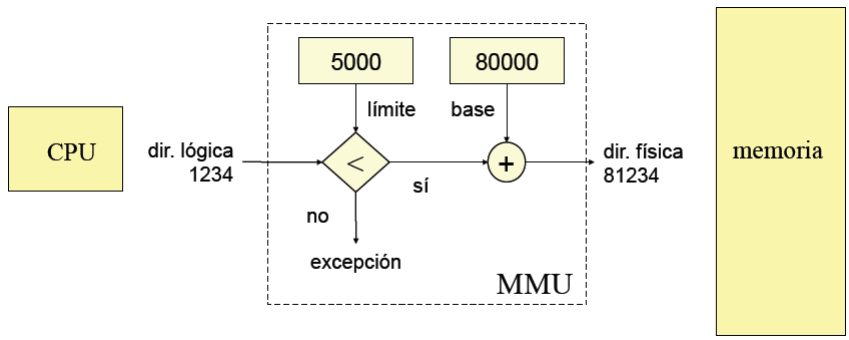
3. **En tiempo de ejecución** (Reubicación dinámica) :
   - Durante la ejecución puede moverse el código de un proceso.
   - El dispositivo que traduce direcciones lógicas a físicas se llama **unidad de manejo de memoria** (MMU, en inglés)
     - El registro base protege al SO




dirección física = registro base + dirección lógica

# Enlace de direcciones

- En el procesador se puede poner un registro límite para evitar que un programa acceda a direcciones fuera de su espacio comprometiendo la privacidad



# Unidades de dirección

- En una arquitectura dada, los sucesivos valores de direcciones designan sucesivas unidades de memoria.
- En muchos ordenadores, la unidad puede ser un byte o una palabra
- Si la unidad es una **palabra**, entonces se puede acceder a una gran suma de memoria utilizando una dirección de un tamaño dado (múltiplo de 2)
- Por otra parte, si la unidad es un **byte**, se pueden direccionar caracteres individuales

# Gestión de memoria. Requisitos

- **Reubicación**
  - El programador no sabe donde se va a colocar un programa en la memoria cuando se ejecute
  - Cuando el programa se está ejecutando, puede ser llevado a disco y vuelto a la memoria principal en un lugar diferente (reubicado)
  - Las referencias de memoria del código se deben poder traducir a direcciones de memoria física que reflejen la ubicación actual en memoria principal

# Gestión de memoria. Requisitos

- **Protección**
  - Los procesos no deben ser capaces de hacer referencia a posiciones de memoria de otro proceso sin permiso
  - Es imposible comprobar las direcciones absolutas en tiempo de compilación. Se debe comprobar en el momento de ejecución
  - El requisito de Protección de la memoria debe ser satisfecha por el procesador (hardware) en lugar del sistema operativo (software)
  - El sistema operativo no puede anticipar todas las referencias de memoria que un programa hará

# Gestión de memoria. Requisitos

- **Compartición**
  - Permitir a varios procesos acceder a la misma porción de la memoria principal
  - Es mejor permitir que cada proceso pueda acceder a la misma copia del programa en lugar de tener su propia copia separada
  - Permitir el acceso controlado a áreas de memoria compartidas sin comprometer la protección.

# Gestión de memoria. Requisitos

- **Organización lógica**
  - Los programas se organizan en **módulos**
  - Los módulos se pueden escribir y compilar de forma independiente
  - Proporcionar diferentes grados de protección a los módulos (de sólo lectura, sólo ejecución)
  - Introducir mecanismos para compartir módulos entre los procesos

# Gestión de memoria. Requisitos

- **Organización física**
  - La memoria disponible para un programa más sus datos puede ser insuficiente
    - El programador podría superponer diferentes módulos (**overlaying**) para asignar la misma región de memoria
  - El programador no sabe cuánto espacio estará disponible
    - Mover información entre memoria principal y secundaria ha de ser responsabilidad del sistema

## Esquemas de asignación de memoria

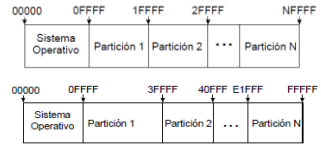
- Contigua: particiones fijas y variables
- Segmentación
- Paginación
- Segmentación paginada

## Esquemas de asignación de memoria

- **Contigua: particiones fijas y variables**
- Segmentación
- Paginación
- Segmentación paginada

# Particionamiento de memoria

- La memoria está dividida de antemano en espacios (Particiones).
- Un proceso necesita ejecutarse ⇒ Se le asigna una partición
- Cada partición puede contener un único proceso.
- Pueden ser:
  - Particiones **Fijas**:
    - Todas el mismo tamaño.
    - Con diferentes tamaños.
  - Particiones **Variables**.




- Tabla de descripción de particiones (TDP)

# Particionamiento de memoria

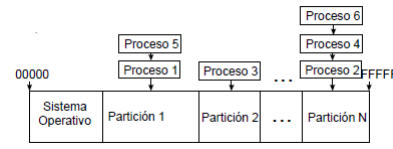
- **Particiones fijas de igual tamaño.**
  - Cualquier proceso cuyo tamaño <= que el tamaño de partición puede cargarse en cualquier partición disponible
  - Hay una tabla para indicar particiones ocupadas y libres.
  - Hay una cola con procesos que quieren utilizar memoria y ejecutarse.
  - Si todas las particiones están llenas y ningún proceso en estado Listo o Ejecutando ⇒ se lleva un proceso a disco para dejar espacio a un nuevo proceso (*swap*)

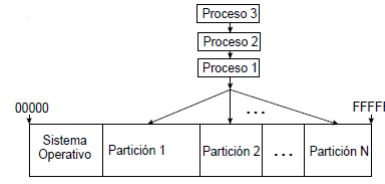
# Particionamiento de memoria

- **Inconvenientes**
  - Nivel de multiprogramación limitado por número de particiones.
  - Si un programa no cabe en una partición ⇒ el programador debe diseñarlo con superposiciones
  - Cualquier programa por pequeño que sea ocupa una partición entera ⇒ fragmentación interna.

# Particionamiento de memoria

- **Particiones fijas de diferentes tamaños**
  - Una cola de procesos para cada partición
    - Cada proceso se asigna a una cola en función de su tamaño
    - Se minimiza la fragmentación interna
  - Una sola cola de procesos
    - Al cargar el proceso se selecciona la partición más pequeña disponible en la que quepa
    - Si todas las particiones están ocupadas se envía un proceso a disco
      - El que ocupe la partición más pequeña que pueda albergar al nuevo
      - Otros factores: prioridad, preferencia bloqueados/listos



# Particionamiento de memoria

- **Ejercicio**
  - En un sistema de particionamiento fijo, con particiones de 100K 500K, 200K, 300K y 600K (en este orden) y que todas están disponibles, mostrar el estado de la memoria después de las siguientes solicitudes: 212K, 417K, 112K y 350K. Indicar también la cantidad de fragmentación interna que se produce.

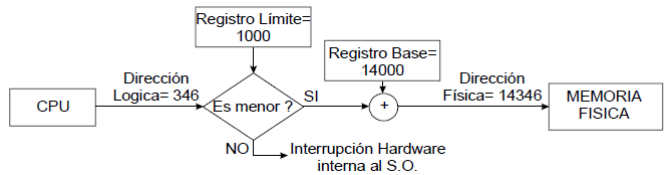
# Particionamiento de memoria

- **Solución**
  - 100K
  - La partición de 500K alberga 417K
  - La partición de 200K alberga 112K
  - La partición de 300K alberga 212K
  - La partición de 600K alberga 350K

  - La fragmentación interna que se produce es (500-417)+(200-112)+(300-212)+(600-350)=509K

# Particionamiento de memoria

- Necesidad de protección: (en sistemas multiprogramados)
  - Un proceso no acceda al área de memoria del otro.
  - Si la reubicación es dinámica puede usarse registros base-límite.
    - El registro base almacena la dirección de la partición en donde se cargó el trabajo
    - El registro límite almacena la longitud de la partición




# Particionamiento de memoria

- **Particiones variables (particionamiento dinámico)**
  - Particiones de longitud y nº variable
  - Inicialmente: Toda la memoria (salvo partición del S.O.) disponible para procesos, como si fuese un gran hueco.
  - Llega un proceso:
    - Se introduce en un hueco libre.
    - El espacio no ocupado será un nuevo hueco.
  - Cada zona de memoria ocupada -> una partición.
  - Un proceso termina:
    - Libera su zona de memoria.
    - Se convierte en un hueco.
    - Dicho hueco se fusiona con los adyacentes.
  - Se conserva una tabla de partes de memoria ocupadas y libres y la cola de entrada de procesos en memoria.

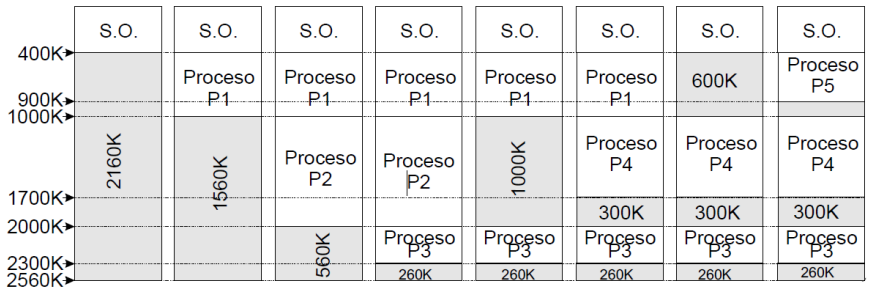
# Particionamiento de memoria

- **Particiones variables.**

  Los procesos se cargan en memoria, compiten por la CPU y al acabar liberan la memoria

| Proceso | Memoria Requerida |
|---|---|
| $P_1$ | 600 K |
| $P_2$ | 1000 K |
| $P_3$ | 300 K |

| Proceso | Memoria Requerida |
|---|---|
| $P_4$ | 700 K |
| $P_5$ | 500 K |

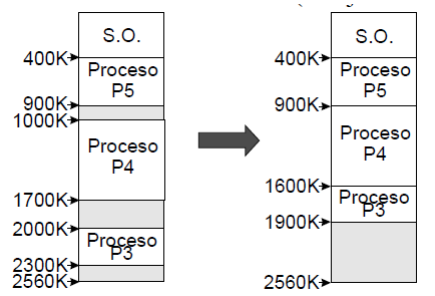



# Particionamiento de memoria

- **Particiones variables.**
  - Al proceso se le asigna exactamente tanta memoria como requiere, no más (no hay fragmentación interna)
  - Se van formando huecos en la memoria.
    - Un proceso quiere ejecutarse, hay una partición libre, pero de menor tamaño que el proceso (fragmentación externa)

# Particionamiento de memoria

- **Particiones variables.**
  - Se debe usar compactación para desplazar los procesos para que queden contiguos y toda la memoria libre esté unida en un bloque
    - Consume tiempo

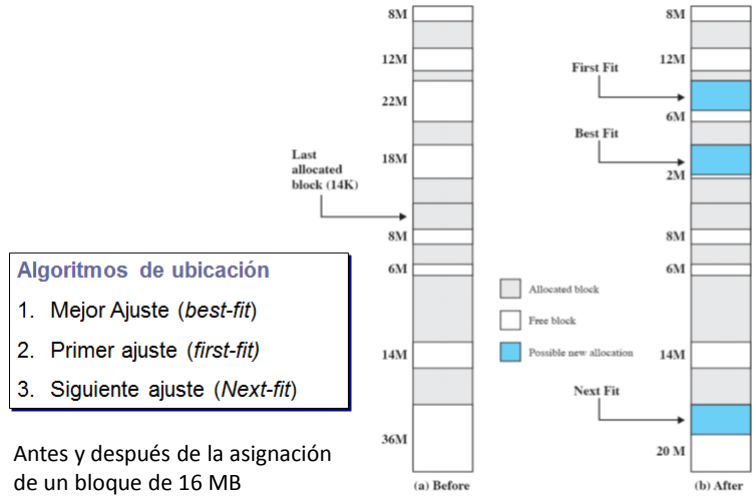



# Particionamiento de memoria

- **Particiones variables.**
  - Algoritmos para intercambiar un proceso (políticas de ubicación)
    - Primer ajuste
      - **Escoge el primer hueco libre de tamaño suficiente.**
      - Es el más rápido y sencillo
      - Puede tener muchos procesos cargados en el extremo delantero de la memoria entre los que se ha de encontrar un bloque libre
    - Mejor ajuste
      - **Hueco que mejor se ajuste a su tamaño**
      - Peor desempeño global
      - Mucha fragmentación externa ⇒ compactación frecuente
    - Siguiente ajuste
      - **Hueco suficientemente grande desde el último que ha sido asignado**
      - El mayor bloque de memoria, que se encuentra al final, se divide rápidamente en bloques más pequeños (fragmentación externa) ⇒ compactación frecuente

# Particionamiento de memoria

- **Particiones variables.**




**Algoritmos de ubicación**

1. Mejor Ajuste (*best-fit*)
2. Primer ajuste (*first-fit*)
3. Siguiente ajuste (*Next-fit*)

Antes y después de la asignación de un bloque de 16 MB

# Particionamiento de memoria

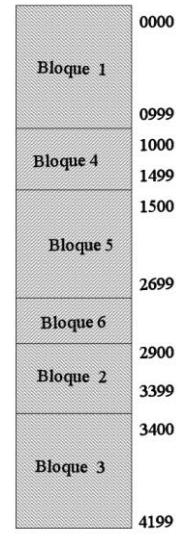
- **Ejercicio**
  - Sea un sistema gestionado con un mecanismo de particiones variables en el que la memoria física tiene 4200 palabras. En un instante la memoria está ocupada por 3 bloques de código/datos de la forma:

| Dirección inicial | longitud |
|---|---|
| 1000 | 1000 |
| 2900 | 500 |
| 3400 | 800 |

  - La estrategia utilizada cuando se carga un nuevo bloque en memoria es la del **mejor ajuste** en primer lugar. Si falla, se crea un hueco mayor desplazando los bloques en memoria hacia la dirección 0. Esta acción siempre empieza con el bloque actualmente en la dirección de memoria más baja, y prosigue únicamente hasta encontrar un hueco suficiente para el nuevo bloque.
  - A partir de ese momento, hay que cargar tres bloques de 500, 1200 y 200 (en ese orden). Describir el contenido de la memoria una vez satisfechas las peticiones.

# Particionamiento de memoria

- **Ejercicio (solución)**




# Particionamiento de memoria

- **Ejercicio**
  - Sea un sistema gestionado por particiones múltiples de tamaño variable con compactación. En un instante dado, se tiene la siguiente ocupación de la memoria:


  - Se utiliza la técnica del **mejor ajuste**. En la cola de trabajos tenemos en este orden: P4(120K), P5(200K) y P6(80K), los cuales deben ser atendidos en orden FIFO. Suponiendo que no finaliza ningún proceso y tras intentar cargar en memoria todos los procesos que están en la cola.
    a) Indicad cuántas particiones quedan libres y de qué tamaño son.
    b) Si en esta situación se aplica compactación, indicar qué proceso o procesos deberían moverse para que el número de Kbytes manejados fuese el menor posible y quede un único hueco.

# Particionamiento de memoria

- **Ejercicio (solución)**

  **a)** Quedan dos particiones de tamaños 120K y 30 K respectivamente.


  b) Debería moverse el proceso P4 al hueco de 120K, con lo cual quedaría un solo hueco de 150K.

# Particionamiento de memoria

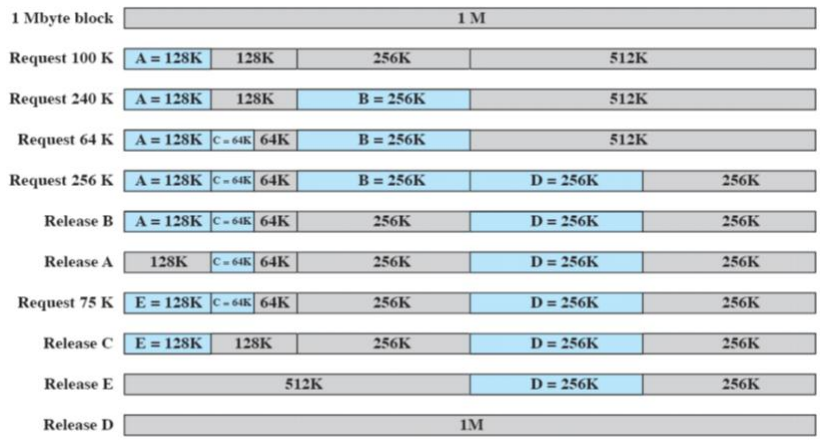
- **Sistema de colegas (buddy system)**
  - Presenta un equilibrio razonable para superar las desventajas de los esquemas de partición fija y variable
  - Los bloques de memoria disponibles son de tamaño $2^k$, con $L \leq K \leq U$, donde:
    - $2^L$= tamaño de bloque más pequeño asignable
    - $2^U$ = tamaño de bloque más grande asignable
  - Inicialmente, la memoria disponible se trata como un bloque de tamaño máximo $2^U$
  - Mantiene en todo momento una lista de huecos (bloques no asignados) para cada tamaño $2^i$

# Particionamiento de memoria

- **Sistema de colegas** (cont.)
  - Supongamos que se realiza una solicitud de tamaño S
    - Si $2^{U-1} < S \leq 2^{U} \Rightarrow$ se le asigna el bloque entero $2^{U}$
    - En caso contrario:
      - Se divide el bloque en dos colegas de igual tamaño $2^{U-1}$
      - Si $2^{U-2} < S \leq 2^{U-1} \Rightarrow$ se le asigna uno de los colegas $2^{U-1}$
      - En caso contrario, uno de los colegas se divide por la mitad nuevamente, de tamaño $2^{U-2}$
      - El proceso continúa hasta que el bloque más pequeño sea ≥ S
  - Cuando una pareja de colegas de la lista $i$ pasa a estar libre, se les elimina de esa lista y se unen en un solo bloque de la lista $(i + 1)$

# Particionamiento de memoria

- **Sistema de colegas** (cont.)



# Particionamiento de memoria

- **Sistema de colegas** (cont.)
  - Dada una solicitud de tamaño k, con $2^{i-1} < k \leq 2^i$, para encontrar un hueco de tamaño 2i:

```
void conseguir_hueco(int i){
   if (i == (U+1)) return ERROR;
   if (vacia(lista_i)) {
      conseguir_hueco(i+1);
      dividir_hueco_en_colegas;
      poner_colegas_en_lista_i;
   }
   coger_primer_hueco_de_la_lista_i;
}
```

  - Se usa en sistemas paralelos para asignar y liberar programas en paralelo
  - UNIX usa una forma modificada del sistema de colegas para la asignación de memoria en el núcleo

# Particionamiento de memoria

- **Ejercicio**
  - Un bloque de memoria de 1Mbyte se asigna utilizando el sistema buddy, mostrar gráficamente las divisiones y fusiones de bloques que se van generando para las siguientes peticiones:
    - Solicitar 70
    - Solicitar 35
    - Solicitar 80
    - Liberar A
    - Solicitar 60
    - Liberar B
    - Liberar D
    - Liberar C

# Particionamiento de memoria

- **Ejercicio (solución)**

| | | | | | | |
|---|---|---|---|---|---|---|
| Request 70 | A | 128 | | 256 | | 512 |
| Request 35 | A | B | 64 | 256 | | 512 |
| Request 80 | A | B | 64 | C | 128 | 512 |
| Return A | 128 | B | 64 | C | 128 | 512 |
| Request 60 | 128 | B | D | C | 128 | 512 |
| Return B | 128 | 64 | D | C | 128 | 512 |
| Return D | 256 | | | C | 128 | 512 |
| Return C | 1024 | | | | | |

# Esquemas de asignación de memoria

- Contigua: particiones fijas y variables
- **Segmentación**
- Paginación
- Segmentación paginada

# Segmentación

- Un programa se puede descomponer en varios **segmentos** de memoria (código, datos, pila...)
- Con el *hardware* adecuado, podemos ubicar esos segmentos en zonas de memoria no contiguas.
- El compilador tiene que generar código que haga referencias a direcciones de memoria con dos componentes: <segmento, desplazamiento>
- Similar al particionamiento dinámico pero
  - Un programa puede ocupar más de una partición
  - Las particiones no necesitan ser contiguas
- No hay fragmentación interna pero sí externa

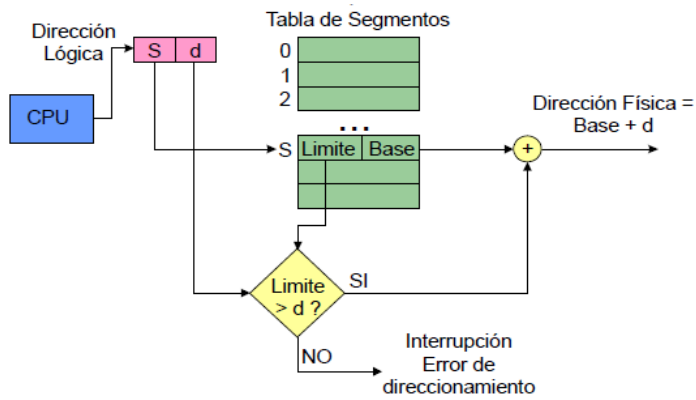
# Segmentación

- Un procesador Intel 8086 utiliza 4 segmentos:
  - Segmento de Código (CS)
  - Segmento de Datos (DS)
  - Segmento de Pila (SS)
  - Segmento extendido (ES)

- Definen áreas de 64 Kb dentro del espacio de direcciones de 1 Mb del 8086. Estas áreas pueden solaparse total o parcialmente.
- No es posible acceder a una posición de memoria no definida por algún segmento: si es preciso, habrá de moverse alguno.
  - Se utilizan registros base y límite para cada segmento

# Segmentación

- Hay una tabla de segmentos por cada proceso
  - Proporciona la dirección inicial de la memoria principal del correspondiente segmento (base)
  - Y la longitud del segmento (límite)




# Segmentación

- **Ejercicio**
  - Hallar la dirección física correspondiente a la dirección lógica (3, 62) dada la siguiente tabla de segmentos de un proceso:

| | Base | Límite |
|---|---|---|
| 0 | 200 | 20 |
| 1 | 50 | 10 |
| 2 | 105 | 49 |
| 3 | 320 | 70 |

- **Solución**
  - (**3**, 62) ⇒ Base= 320 y Límite = 70
  - 70 > 62 ⇒ DF = 320 + 62 = 382

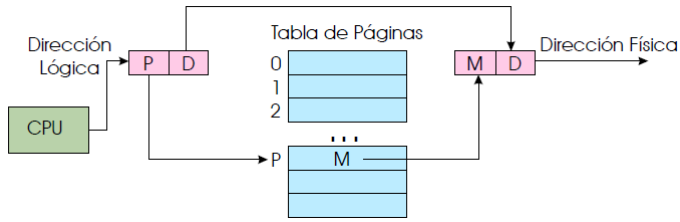
## Esquemas de asignación de memoria

- Contigua: particiones fijas y variables
- Segmentación
- **Paginación**
- Segmentación paginada

## Paginación

- Resuelve el problema de fragmentación externa
- Permite que la memoria de un proceso no sea contigua.
- La memoria **física**: la dividimos en bloques de **tamaño fijo**: ***marcos***.
- La memoria **lógica**: la dividimos en bloques llamados: ***páginas*** (de igual tamaño que el marco)
- Las páginas de un proceso se cargan en los marcos de la memoria principal que estén disponibles
  - Tenemos "trozos" del proceso allí donde la memoria está disponible.
- Se establece una tabla de páginas (TP)

# Paginación




- La dirección lógica generada consta de dos partes:
  - Nº de Pagina (P).
  - Desplazamiento dentro de la página (D).
- La tabla de páginas:
  - Contiene la dirección base en memoria física
  - Permite establecer una correspondencia entre el nº de página y un nº de marco de memoria física.
- La dirección física es el nº de marco (M) y el desplazamiento (D).

# Paginación

- Es similar al particionamiento fijo pero
  - Las particiones son bastante más pequeñas
  - Un programa puede ocupar más de una partición
  - Las particiones no necesitan ser contiguas
- El tamaño de página es una potencia de 2 (y múltiplo del tamaño de disco: 1KB-16KB)

# Paginación

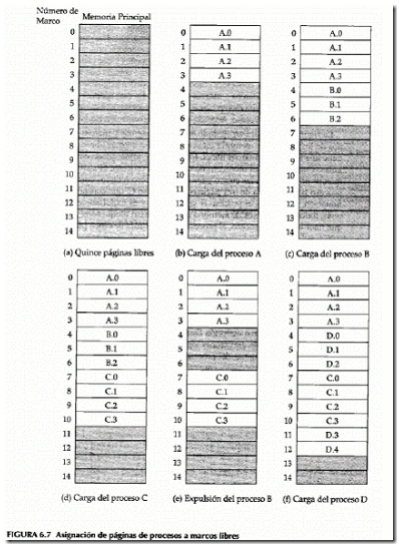



FIGURA 6.7 Asignación de páginas de procesos a marcos libres

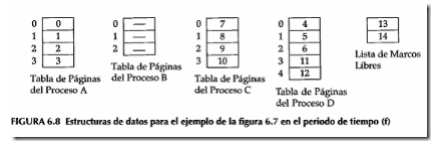


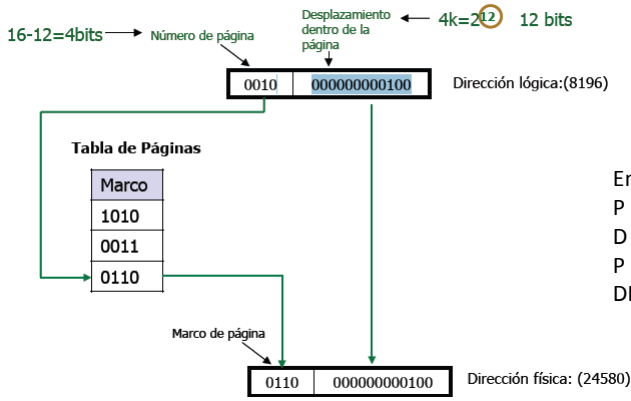

FIGURA 6.8 Estructuras de datos para el ejemplo de la figura 6.7 en el periodo de tiempo (f)

# Paginación

- Traducción de direcciones (en binario)
  - Direcciones de n+m bits (n:P, m:D)
  - Se extrae el nº de página (P) extrayendo los n bits de la izqda de la dirección lógica
  - Se utiliza P como índice de la TP del proceso para encontrar en nº de marco M
  - Se concatena **M+D**
- Traducción de direcciones (en decimal)
  - P = DL **/** tam_pag
  - D = DL **mod** tam_pag
  - Se utiliza P como índice de la TP del proceso para encontrar en nº de marco M
  - **DF = marco * tam_pag + D**

# Paginación

- Traducción de direcciones:
  - Ejemplo: direcciones de 16 bits y tamaño de página 4KB




En decimal:
P = 8196 / 4096 = 2
D = 8196 mod 4096 = 4
P 2 ⇒ M 6
DF = 6 * 4096 + 4 = 24580

# Paginación

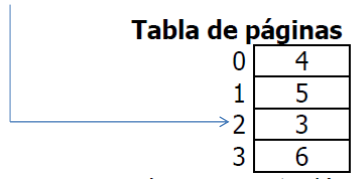
- **Ejercicio**
  - Considera un esquema de paginación, con un tamaño de página de 256 palabras. Las direcciones lógicas generadas son 530, 1046. Calcular las direcciones físicas. El contenido de la tabla de páginas es el siguiente:

**Tabla de páginas**

| | |
|---|---|
| 0 | 4 |
| 1 | 5 |
| 2 | 3 |
| 3 | 6 |

# Paginación

- **Ejercicio (solución)**
  a) 530
     - 530 = 1000010010
     - 256 = $2^8$ ⇒ 8 bits de desplazam.
     - 10 | 00010010




     - 11 | 00010010 = 786

  b) 1046
     - 1046 = 100 | 00010110
     - ERROR

a) 530
- P = 530/256 = 2
- D = 530 mod 256 = 18
- P 2 ⇒ M 3
- DF = 3*256+18=786

a) 1046
- P = 1046/256=4
- ERROR

# Paginación

- **Ejercicio**
  - Considera los 4 sistemas con gestión de memoria mediante paginación A, B, C y D siguientes:

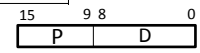
| Sistema | A | B | C | D |
|---|---|---|---|---|
| Tamaño de página (en palabras) | 512 | 512 | 1024 | 1024 |
| Tamaño de palabra (en bits) | 16 | 32 | 16 | 32 |

  - Asumiendo que sólo hay una tabla de páginas para todo el sistema y que tanto la dirección lógica como el descriptor de página ocupa una palabra (de 16 o 32 bits, según el caso), determina para cada sistema:
    a) El tamaño de la tabla de páginas (número de entradas).
    b) El tamaño de la memoria lógica (número de páginas).

# Paginación

- **Ejercicio (solución)**

| Sistema | A | B | C | D |
|---|---|---|---|---|
| Tamaño de página (en palabras) | 512 | 512 | 1024 | 1024 |
| Tamaño de palabra (en bits) | 16 | 32 | 16 | 32 |
| Tamaño de la tabla de páginas (en descriptores) | 128 | 8 M | 64 | 4M |
| Tamaño de memoria lógica (en páginas) | 128 | 8M | 64 | 4M |




A. $512 = 2^9$; $16 - 9 = 7$; $2^7 = 128$
B. $512 = 2^9$; $32 - 9 = 23$; $2^{23} = 8388608$
C. $1024 = 2^{10}$; $16 - 10 = 6$; $2^6 = 64$
D. $1024 = 2^{10}$; $32 - 10 = 22$; $2^{22} = 4194304$

# Paginación

- **Ejercicio**
  - Considera un espacio de direcciones lógicas de 8 páginas de 1024 bytes cada una, mapeadas en un espacio físico de memoria de 32 marcos
    a) Cuántos bits tiene la dirección lógica
    b) Cuántos bits tiene la dirección física

# Paginación

- **Solución**

  a) 8 * 1024 = 8192 = $2^{13}$
  Se necesitan 13 bits para las direcciones lógicas
  El tamaño de página es 1024 = $2^{10}$, por tanto necesitamos 10 bits para el desplazamiento, y 3 bits para direccionar las 8 páginas.

  b) 32 * 1024 = 32.768 = $2^{15}$
  Se necesitan 15 bits para las direcciones físicas
  El tamaño de marco ha de ser el mismo que el de página, por tanto necesitamos 10 bits para el desplazamiento. Para direccionar 32 marcos necesitamos 5 bits.

# Paginación

- La tabla de páginas reside en memoria principal
- Se tiene un registro base de tabla de páginas (RBTP) para saber a qué TP hay que acceder (hay una por cada proceso)
- Para cambiar de tabla de páginas ⇒ Basta cambiar de registro base.
  - Menor tiempo de cambio de contexto
- Hay que proteger el acceso a la tabla de páginas
  - Registro límite de tablas de página (RLTP)
- Se requieren 2 accesos a memoria, uno para acceder a la TP y otro a la dirección física
  - Solución: *caché* de traducciones → **TLB**

# Paginación

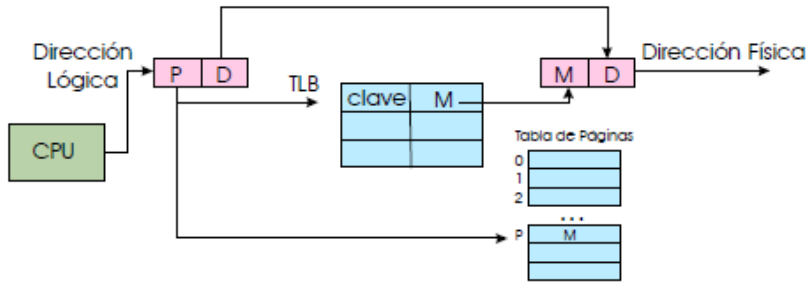
- **TLB** *(Translation Lookaside Buffer)*
  - Reduce las pérdidas de tiempo de acceso a RAM
  - Las entradas contienen en nº de página (clave) y su marco, de las últimas páginas accedidas
  - Es una memoria direccionable por contenido o memoria asociativa (va dentro del procesador)
  - Se compara el valor de la página deseada con todas las claves (en paralelo)
    - Si la clave está: Proporciona el número de marco asociado.
    - Si no está: Se accede a la tabla de páginas de memoria.

# Paginación

- **TLB**
  - Cuando se pasa de un proceso a otro ocurre un cambio de contexto y se han de invalidar las entradas del proceso anterior en el TLB (para evitar problema seguridad)
    - No se invalidan todas, sólo las que necesitamos para el nuevo proceso para que cuando retornemos al anterior se puedan reutilizar las suyas.
      - Habrá que etiquetar cada entrada a qué proceso pertenece
  - Los procesadores suelen reservar 8 entradas del TLB para el SO (si no las llamadas al kernel continuamente darían fallo de TLB)
  - Se puede mejorar la tasa de aciertos del TLB aumentando el tamaño de página

# Paginación

- **TLB**




# Memoria Virtual

# Memoria virtual

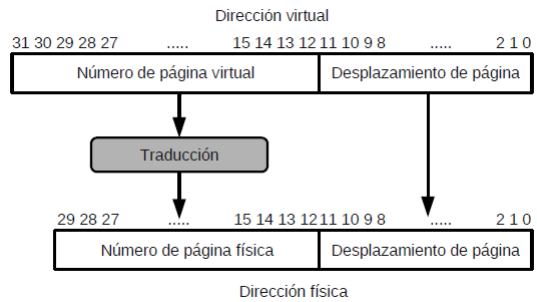
- Permite separar la memoria lógica del usuario de la memoria física.
- Un proceso en ejecución no tiene porqué encontrarse totalmente en memoria principal (sólo parte).
  - La parte del proceso que se encuentra realmente en memoria se denomina **conjunto residente** del proceso
- Ahora un proceso puede ser mayor que la memoria física.
- Permite transferencia de información entre memoria principal y secundaria (2 niveles consecutivos de la jerarquía de memoria).
- Usa un dispositivo de almacenamiento secundario (disco) como dispositivo de intercambio.

# Memoria virtual

- Permite ejecutar programas o procesar datos cuyo tamaño excede el espacio de memoria disponible,
- y/o tener en ejecución varios programas a la vez,
- usando un esquema de segmentación / paginación por demanda, es decir, manteniendo en memoria principal sólo la porción del programa que se este ejecutando

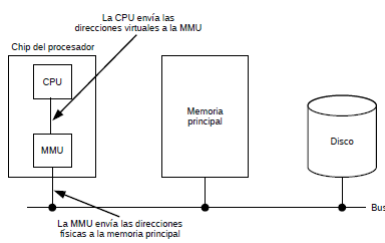
# Memoria virtual

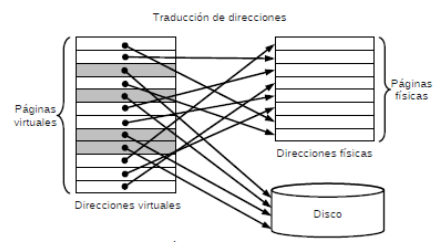
- Ejemplo:
  - Desplazamiento dentro de la página de 12 bits ⇒ Tamaño de la página es $2^{12}$ bytes = 4 KB.
  - Nº de página física de 18 bits ⇒ $2^{18}$ páginas físicas.
  - La memoria principal es de $2^{30}$ bytes = 1 GB.
  - La memoria virtual es de $2^{32}$ bytes = 4 GB.




# Memoria virtual

- Traducción de direcciones:
  - La CPU genera direcciones virtuales.
  - La MMU las traduce a direcciones físicas.
  - A memoria se accede con direcciones físicas.

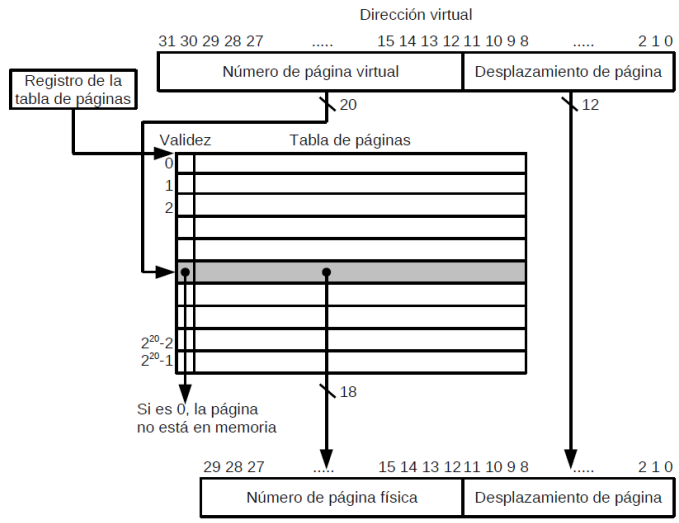


# Paginación por demanda

- La estructura de la tabla de páginas la define el procesador
  - Reside en memoria.
  - Cada programa tiene su propia tabla de páginas.
  - Registro de la tabla de páginas: apunta al inicio de la tabla de páginas del programa que esté en funcionamiento.
  - La Tabla de páginas se indexa con el número de página virtual.
  - Contiene el correspondiente número de página física (marco)
  - Podemos guardar bits adicionales:
    - rwx: permisos. Los mete el SO, los verifica el procesador
    - p: presencia *(bit de validez)*. Lo mete el SO, lo verifica el procesador cuando va a acceder, si no está da un fallo de página
    - a: *access bit*. Lo mete el procesador cada vez que accede a una página
    - d: *dirty bit (bit de modificación)*. Lo mete el procesador cuando el contenido de la página ha sido alterado.

# Paginación por demanda



# Paginación por demanda

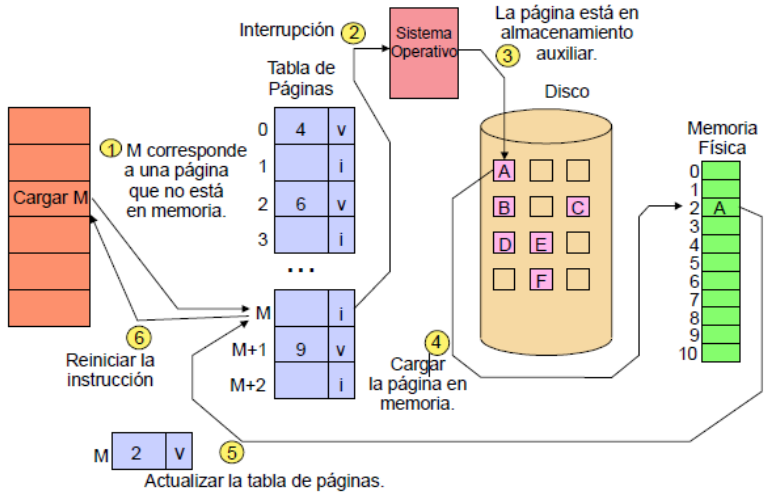
- Supongamos que se tiene que traer un nuevo proceso a memoria
  - El SO comienza trayendo la porción inicial del programa y la de datos sobre la cual acceden las primeras instrucciones.
  - El resto permanece en almacenamiento secundario.
  - Se lee el bit de presencia de la tabla de paginas:

# Paginación por demanda

1: página cargada en memoria principal (bit válido).
- la ejecución prosigue normalmente

0: página no cargada (bit inválido).
- Ocurre una interrupción o **fallo de página** ⇒ Control al SO
- El SO coloca el proceso en estado de Bloqueado
- Para reanudar el proceso bloqueado, el SO ha de traer a memoria principal la porción del proceso que contiene la dirección lógica solicitada ⇒ **petición E/S** (lectura a disco)
- Mientras el SO puede activar la ejecución de otro proceso
- Cuando la porción ya está en memoria principal se genera otra interrupción de E/S y el SO toma el control y se actualiza la TP
- El SO coloca el proceso afectado al estado de Listo

# Paginación por demanda



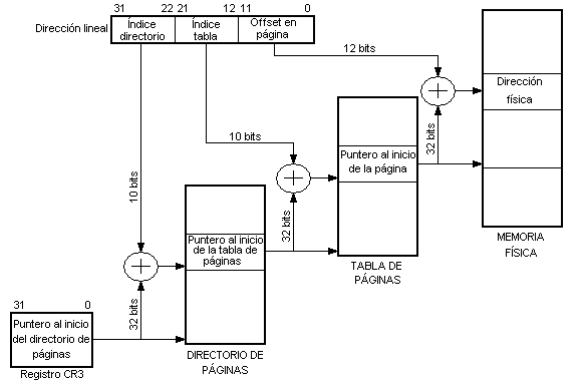

# Tablas de páginas de varios niveles

- Para organizar las tablas de página de gran tamaño algunos procesadores utilizan un esquema de varios niveles.
- Cada tabla puede estar en cualquier página y se puede utilizar memoria secundaria
- Se necesita guardar en memoria principal al menos una página de cada nivel

- **Intel** utiliza sistemas de paginación multinivel

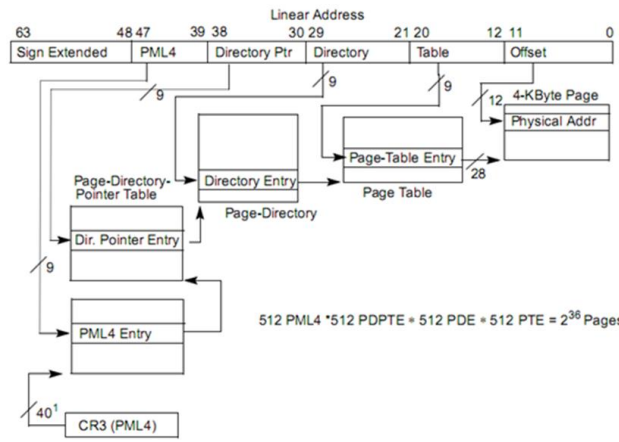
# Ejemplo procesador x86-32: arquitectura 2 niveles de páginas

- El registro CR3 apunta a un directorio de páginas de 4KB, que está dividido en 1024 ($2^{10}$) entradas de 4B, que apuntan a 1024 tablas de páginas
- $2^{10}$ entradas de directorio * $2^{10}$ entradas de tablas de páginas = $2^{20}$ páginas para cada proceso




# Ejemplo procesador x86-64: arquitectura 4 niveles de páginas

- Direccionamiento de 48 bits
- $2^9$ PML4E * $2^9$ PDPTE * $2^9$ PDE * $2^9$ PTE = $2^{36}$ páginas para cada proceso
- 1 proceso puede llegar a tener $2^{27}$ tablas de páginas

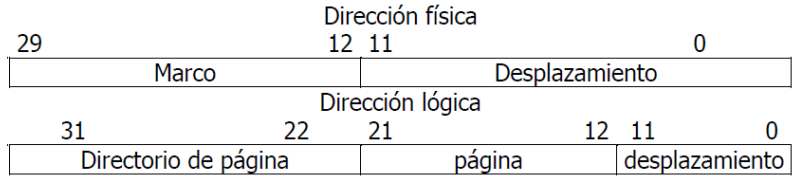


# Tablas de páginas de varios niveles

- **Ejercicio**
  - Se tiene un sistema de memoria con paginación a dos niveles que permite agrupar las páginas en "directorios de páginas".
  - Cada directorio de páginas puede contener 1024 páginas.
  - Los espacios de direcciones lógicas de este sistema son de 4GB, el tamaño de página es de 4KB y el del descriptor de página es de 4 bytes.
  - El espacio de direcciones físicas puede tener hasta 1GB.
  - Describir la estructura de las direcciones lógicas y de las direcciones físicas de este sistema de memoria virtual.

# Tablas de páginas de varios niveles

- **Ejercicio (solución)**
  - Directorios de 1024 páginas: $1024 = 2^{10} \Rightarrow$ 10 bits para el índice de directorios
  - Direcciones lógicas de 4GB: $4.294.967.296 = 2^{32} \Rightarrow$ 32 bits
  - Tamaño de página de 4KB: $4096 = 2^{12} \Rightarrow$ 12 bits de desplazamiento
  - Direcciones físicas de 1GB: $1.073.741.824 = 2^{30} \Rightarrow$ 30 bits



# Tablas de páginas de varios niveles

- **Ejercicio**
  - Sea un sistema con Memoria virtual que utiliza paginación multinivel con 2 niveles de paginación.
  - El tamaño de la dirección física es de 13 bits , el tamaño de marco es de 1024 bytes y el número máximo de entradas de las tablas de páginas de ambos niveles es de 4 descriptores de página en cada nivel.
    a) Indique cada uno de los campos de la dirección lógica y física de este sistema, así como el número de bits de dichos campos.
- **Solución**

a) Dirección Lógica

| Bit 13 | | Bit 0 |
|---|---|---|
| 1er nivel | 2º nivel | Desplaz. Página |
| 2 bits | 2 bits | 10 bits |

Dirección Física

| Bit 12 | Bit 0 |
|---|---|
| Marco | Desplazamiento |
| 3 bits | 10 bits |

# Tablas de páginas de varios niveles

- **Ejercicio (cont.)**

b) En la actualidad en dicho sistema se están ejecutando dos procesos P1 y P2, y el contenido de la memoria principal es el que se muestra a continuación:

Memoria Principal

| Núm. marco | Contenido (proceso,página) |
|---|---|
| 0 | (P1, 4) |
| 1 | (P2, 8) |
| 2 | (P1, 7) |
| 3 | (P2, 10) |
| 4 | (P2, 11) |
| 5 | (P1, 5) |
| 6 | (P1, 6) |
| 7 | Tabla pág. |

b1) Dada la tabla de páginas del primer nivel del proceso P1, indique el contenido de sus tablas de páginas del 2º nivel
Indique también el descriptor de página de 1er nivel que sería necesario utilizar con el fin de acceder a las páginas que están ubicadas en memoria.
Cada descriptor de páginas ocupa 2 bytes

Proceso P1
Tabla de páginas
1er nivel

| |
|---|
| 8000 |
| 8008 |
| 8016 |
| 8024 |

# Tablas de páginas de varios niveles

- **Solución**

Memoria Principal

| Núm. marco | Contenido (proceso,página) |
|---|---|
| 0 | (P1, 4) |
| 1 | (P2, 8) |
| 2 | (P1, 7) |
| 3 | (P2, 10) |
| 4 | (P2, 11) |
| 5 | (P1, 5) |
| 6 | (P1, 6) |
| 7 | Tabla pág. |

| Valor del descriptor de página |
|---|
| 8008 |

Proceso P1
Tabla de páginas
1er nivel

| |
|---|
| 8000 |
| 8008 |
| 8016 |
| 8024 |

Proceso P1
Tablas de páginas 2º nivel

| | marco | Bit validez |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 5 | 1 |
| 2 | 6 | 1 |
| 3 | 2 | 1 |

# Tablas de páginas de varios niveles

- **Ejercicio (cont.)**

Memoria Principal

| Núm. marco | Contenido (proceso,página) |
|---|---|
| 0 | (P1, 4) |
| 1 | (P2, 8) |
| 2 | (P1, 7) |
| 3 | (P2, 10) |
| 4 | (P2, 11) |
| 5 | (P1, 5) |
| 6 | (P1, 6) |
| 7 | Tabla pág. |

b2) Dada la tabla de páginas del primer nivel del proceso P2, indique el contenido de sus tablas de páginas del 2º nivel
Indique también el descriptor de página de 1er nivel que sería necesario utilizar con el fin de acceder a las páginas que están ubicadas en memoria.
Cada descriptor de páginas ocupa 2 bytes

Proceso P2
Tabla de páginas
1er nivel

| |
|---|
| 8032 |
| 8040 |
| 8048 |
| 8056 |

# Tablas de páginas de varios niveles

- **Solución**

Memoria Principal

| Núm. marco | Contenido (proceso,página) |
|---|---|
| 0 | (P1, 4) |
| 1 | (P2, 8) |
| 2 | (P1, 7) |
| 3 | (P2, 10) |
| 4 | (P2, 11) |
| 5 | (P1, 5) |
| 6 | (P1, 6) |
| 7 | Tabla pág. |

b2)

Proceso P2
Tabla de páginas
1er nivel

| |
|---|
| 8032 |
| 8040 |
| 8048 |
| 8056 |

Proceso P2
Tablas de páginas 2º nivel

| | marco | Bit validez |
|---|---|---|
| 0 | 1 | 1 |
| 1 | - | - |
| 2 | 3 | 1 |
| 3 | 4 | 1 |

| | | |
|---|---|---|
| 0 | - | - |
| 1 | - | - |
| 2 | - | - |
| 3 | - | - |

**Valor del descriptor**: 8048

**Valor del descriptor**: 8040

# Tablas de páginas de varios niveles

- **Ejercicio (cont.)**
  - c) Ambos procesos utilizan los mismos datos de entrada, sobre los que únicamente acceden para lectura. Dichos datos corresponden a los contenidos de las páginas 7 para ambos procesos P1 y P2. Indique como se habrían de modificar los contenidos de las tablas de páginas del apartado anterior para que sea posible la compartición de los datos de entrada.

Memoria Principal

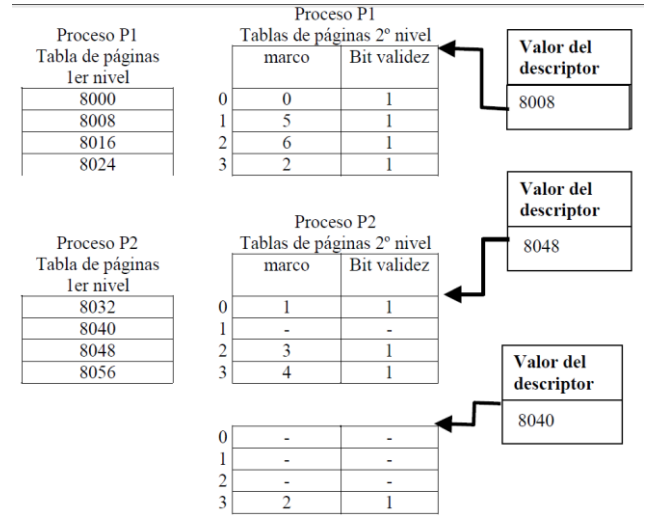
| Núm. marco | Contenido (proceso,página) |
|---|---|
| 0 | (P1, 4) |
| 1 | (P2, 8) |
| 2 | (P1, 7) |
| 3 | (P2, 10) |
| 4 | (P2, 11) |
| 5 | (P1, 5) |
| 6 | (P1, 6) |
| 7 | Tabla pág. |

# Tablas de páginas de varios niveles

- **Solución**

Memoria Principal

| Núm. marco | Contenido (proceso,página) |
|---|---|
| 0 | (P1, 4) |
| 1 | (P2, 8) |
| 2 | (P1, 7) |
| 3 | (P2, 10) |
| 4 | (P2, 11) |
| 5 | (P1, 5) |
| 6 | (P1, 6) |
| 7 | Tabla pág. |




# Principios de localidad

- Temporal: si he accedido a una región de memoria, alta probabilidad de que acceda de nuevo en poco tiempo
- Espacial: si he accedido hace poco a una región de memoria cercana, alta probabilidad de que acceda esta (y menor de una lejana)

# Paginación con TLB

- Cada lectura/escritura a memoria provoca realmente dos accesos a memoria:
  1. Acceso a la tabla de páginas para traducir la dirección virtual a dirección física.
  2. Con la dirección física obtenida, acceder físicamente a la palabra solicitada.
- Solución: **TLB**
  - caché especial que guarda las traducciones nº de página virtual ⇒ nº de página física más frecuentemente usadas
  - Aprovecha la localidad de las referencias a la tabla de páginas.
  - Cuando un programa accede a una palabra que pertenece a una página virtual X, es muy probable que próximamente acceda a palabras de direcciones cercanas y que, por tanto, pertenezcan a la misma página X.

# Paginación con TLB

- En cada referencia a memoria:
  1. La MMU descompone la dirección virtual en NPV + desplazamiento
  2. Se busca el número de página virtual en el TLB.
     - Si hay acierto de TLB, se salta al paso 6.
  3. Como hay un **fallo de TLB**, hay que ir a la tabla de páginas.
     - Si bit de presencia es válido, hay acierto de página. Se salta al paso 5.
  4. Como hay fallo de página, entra a funcionar el S.O.:
     - Coloca la página solicitada en la memoria principal.
     - Actualiza la entrada de la tabla de páginas (el nº de página física y el bit de presencia).
  5. El contenido de la entrada de la tabla de páginas va al TLB. Se actualiza la etiqueta en el TLB.
  6. Usando el TLB: nº de página virtual ⇒ nº de página física.
     - Se actualizan los bits de control de esa página.

# Paginación con TLB

- Los fallos de TLB son más frecuentes que los fallos de páginas, por tener el TLB menos entradas que la tabla de páginas.
  - Hay que seleccionar la entrada del TLB a reemplazar (selección de víctima)
- El TLB suele emplear una estrategia de postescritura.
  - En el paso 6 del algoritmo anterior, la actualización de los bits de control se realiza solamente en el TLB.
- Cuando una entrada en el TLB va a ser sustituida, toda su información debe copiarse en la tabla de páginas (*flush* de TLB)
- Se pueden utilizar TLBs de varios niveles

# Reemplazo de páginas

- Los procesos tienen parte de sus páginas cargadas en memoria.
- En estado estable, la totalidad de los marcos de memoria pueden estar ocupados.
- Si ante un fallo de página no existe un marco libre en memoria principal, el SO puede:
  - Abortar el proceso de usuario (no es una buena solución).
  - Descargar otro proceso y llevarlo a almacenamiento secundario liberando sus marcos (se puede hacer).
  - Reemplazar páginas:
    - usar un algoritmo de reemplazo de páginas para seleccionar un marco víctima que genere el menor número de fallos de página (el marco que tenga menor probabilidad de ser utilizado en un futuro cercano) y liberarlo.

# Reemplazo de páginas

- Cómo reducir el tiempo para resolver los fallos de página:
  - Usar el *dirty bit* en la tabla de páginas
    - Al cargar la página, desde almacenamiento secundario a memoria, el *dirty bit* se pone a 0 (no modificada)
    - Si se escribe en la página el *dirty bit* pasa a 1 (modificado)
    - Si la página es elegida como víctima se mira su *dirty bit*
      - Si la página no ha sido modificada (d=0) no habrá que salvarla
      - Si la página ha sido modificada (d=1) se salvará

# Reemplazo de páginas

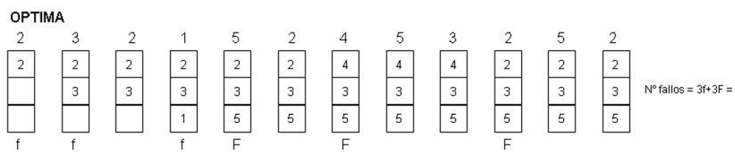
- La tasa de fallos de página dependerá de:
  - Número de páginas de los procesos
  - Número de procesos en memoria
  - Número de marcos disponibles
  - Del algoritmo de reemplazo de páginas (selección de víctima) que se utilice

# Algoritmos de selección de víctima

- Tras un fallo de página se debe elegir **qué** página de memoria se lleva a disco para hacer sitio a la nueva página.
- Objetivo: reemplazar la página con **menor posibilidad** de ser **referenciada** en un futuro cercano => intentar **predecir comportamiento** futuro en función del comportamiento pasado
- Cuanto más elaborada es la política, mayor sobrecarga Sw y Hw

# Algoritmos de selección de víctima

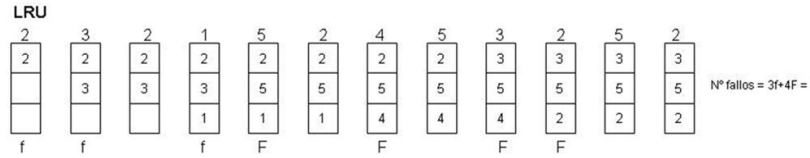
- **Optimo**
  - Se trata de la mejor política posible: **sustituir la página que vaya a tardar más tiempo en ser referenciada en el futuro**
  - No se puede llevar a la práctica en tiempo real, y se utiliza como una referencia teórica para medir la eficiencia de otras políticas en entornos experimentales.




  - Número de fallos de página: **3 (+ 3 iniciales)**

# Algoritmos de selección de víctima

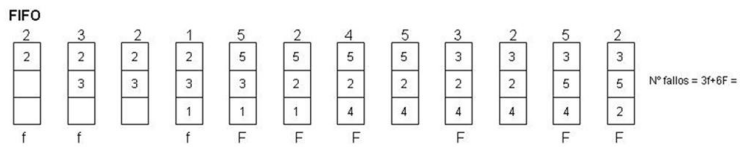
- **LRU** (Least Recently Used)
  - Reemplaza la página de memoria **que no ha sido referenciada desde hace más tiempo.**
  - Debido al principio de localidad espacial, ésta sería la página con menor probabilidad de ser referenciada en un futuro cercano.
    - Se aproxima al óptimo
  - Se podría etiquetar cada página con el momento de su última referencia.




  - Número de fallos de página: **4 (+ 3 iniciales)**

# Algoritmos de selección de víctima

- **FIFO**
  - Se sustituye la página **que lleva más tiempo residente en memoria.**
  - Uso pobre de la localidad temporal
  - Sencillo de implementar




  - Número de fallos de página: **6 (+ 3 iniciales)**

# Algoritmos de selección de víctima

- **FIFO** (cont.)
  - Anomalía de Belady
    - la tasa de fallos pueden aumentar al incrementar el número de marcos asignados

| Page Requests | 3 | 2 | 1 | 0 | 3 | 2 | 4 | 3 | 2 | 1 | 0 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Newest Page | 3 | 2 | 1 | 0 | 3 | 2 | 4 | 4 | 4 | 1 | 0 | 0 |
| | | 3 | 2 | 1 | 0 | 3 | 2 | 2 | 2 | 4 | 1 | 1 |
| Oldest Page | | | 3 | 2 | 1 | 0 | 3 | 3 | 3 | 2 | 4 | 4 |

| Page Requests | 3 | 2 | 1 | 0 | 3 | 2 | 4 | 3 | 2 | 1 | 0 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Newest Page | 3 | 2 | 1 | 0 | 0 | 0 | 4 | 3 | 2 | 1 | 0 | 4 |
| | | 3 | 2 | 1 | 1 | 1 | 0 | 4 | 3 | 2 | 1 | 0 |
| | | | 3 | 2 | 2 | 2 | 1 | 0 | 4 | 3 | 2 | 1 |
| Oldest Page | | | | 3 | 3 | 3 | 2 | 1 | 0 | 4 | 3 | 2 |

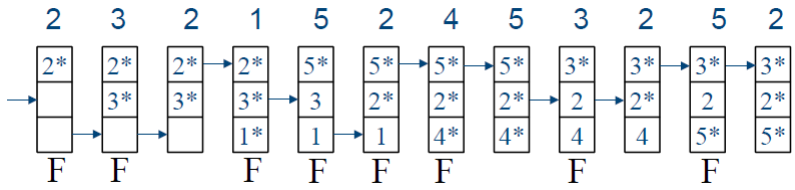
An example of Bélády's anomaly. Using three page frames, 9 page faults occur. Increasing to four page frames causes 10 page faults to occur. Page faults are in red.

# Algoritmos de selección de víctima

- **Reloj (2ª oportunidad)**
  - Mejora de la **FIFO** chequeando si una página ha sido referenciada, haciendo mejor uso de la localidad temporal
    - da una 2ª oportunidad a cada página antes de quitarla de memoria
  - Se utiliza el *bit de acceso* asociado a cada página:
    - 1 cuando se carga la página por primera vez
    - 1 cuando se referencia la página
  - Páginas en cola circular, con puntero a la más antigua
    - Si hay fallo y $a$=0, la página se reemplaza y se avanza el puntero
    - Si hay fallo y $a$=1, se pone $a$=0, avanza y repite lo mismo hasta encontrar página con $a$=0.

# Algoritmos de selección de víctima

- **Reloj** (cont.)




La flecha representa el puntero a la página candidata a ser sustituida, y el asterisco a=1

- Número de fallos de página: **5 (+ 3 iniciales)**

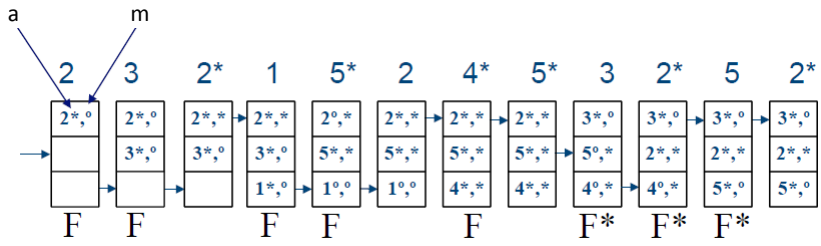
# Algoritmos de selección de víctima

- **NRU** (de Uso No Reciente)
  - Mejora del reloj utilizando bits a y m:
    1. Se busca con a=0 y m=0. No se cambia ningún bit
    2. Si falla paso 1 se busca con a=0 y m=1, se pone a=0 en todos los marcos que se vayan saltando
    3. Si falla paso 2 se repite paso 1 y si hace falta paso 2
  - Ventaja:

    Otorga preferencia para el reemplazo a las páginas que no se han modificado (y que por tanto no necesitan escribirse de nuevo en la memoria secundaria)

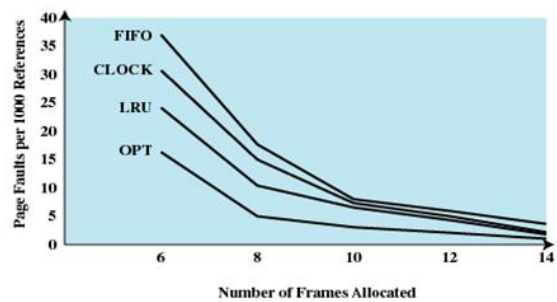
# Algoritmos de selección de víctima

- **NRU** (de Uso No Reciente)




- Número de fallos de página: **5 (+ 3 iniciales)**
- Sólo **3** exigen guardar la página reemplazada en disco

# Comparación algoritmos de selección de víctima




Generador de ejercicios de algoritmos de reemplazo

# Ejercicios

1. Considerando un sistema de memoria virtual con las siguientes solicitudes de páginas:

   0 1 7 0 1 2 0 1 2 3 2 7 1 0

   Obtener la cantidad de fallos de página que se producen con los siguientes algoritmos de reemplazo (utilizando 3 marcos):

   a) óptimo
   b) FIFO
   c) LRU
   d) Reloj

# Ejercicios

a) Optimo

| Cadena de referencias | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 7 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 2 | 7 | 1 | 0 |
| Paso (1-14) | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 7 | 7 | 0 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Pagina que entra | | | | | | | | | | | | | |
| 0 | 1 | 7 | | | 2 | | | | 3 | | 7 | | 0 |
| | | | | | 7 | | | | 0 | | 3 | | 7 |

7 FP

b) FIFO

| Cadena de referencias | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 7 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 2 | 7 | 1 | 0 |
| Paso (1-14) | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 1 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 0 |
| | | 7 | 7 | 7 | 7 | 7 | 1 | 1 | 1 | 1 | 7 | 7 | 7 |
| Pagina que entra | | | | | | | | | | | | | |
| 0 | 1 | 7 | | | 2 | 0 | 1 | | 3 | 2 | 7 | 1 | 0 |
| | | | | | 0 | 1 | 7 | | 2 | 0 | 1 | 3 | 2 |

11 FP

# Ejercicios

c) L R U

| Cadena de referencias | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 7 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 2 | 7 | 1 | 0 |
| Paso (1-14) | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 1 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 | 7 | 7 |
| | | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 |
| Pagina que entra | | | | | | | | | | | | | |
| 0 | 1 | 7 | | | 2 | | | | 3 | | 7 | 1 | 0 |
| | | | | | 7 | | | | 0 | | 1 | 3 | 2 |

8 FP

d) Reloj

11 FP

| * | * | * | | | * | * | * | | * | * | * | * | * |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 7 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 2 | 7 | 1 | 0 |
| >0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 2/1 | 2/1 | >2/1 | 2/1 | 3/1 | 3/1 | >3/1 | 1/1 | 1/1 |
| | 1/1 | 1/1 | 1/1 | 1/1 | >1/0 | 0/1 | 0/1 | 0/1 | >0/0 | 2/1 | 2/1 | >2/0 | 0/1 |
| | | 7/1 | 7/1 | 7/1 | 7/0 | >7/0 | 1/1 | 1/1 | 1/0 | >1/0 | 7/1 | 7/0 | >7/0 |

# Ejercicios

2. Un sistema de memoria virtual con paginación por demanda tiene un tamaño de página de 512 palabras, una memoria virtual de 16 páginas y una memoria física de 4 marcos. El contenido actual de la memoria es:

| Nº marco | |
|---|---|
| 0 | Pág. 4 del proceso P |
| 1 | Pág. 9 del proceso P |
| 2 | Pág. 5 del proceso P |
| 3 | Pág. 1 del proceso P |

a) Mostrar el contenido de la tabla de páginas

| | Tabla de páginas | |
|---|---|---|
| | Marco | Bit de validez |
| 0 | - | 0 |
| 1 | 3 | 1 |
| 2 | - | 0 |
| 3 | - | 0 |
| 4 | 0 | 1 |
| 5 | 2 | 1 |
| 6 | - | 0 |
| 7 | - | 0 |

| | Tabla de páginas | |
|---|---|---|
| | Marco | Bit de validez |
| 8 | - | 0 |
| 9 | 1 | 1 |
| 10 | - | 0 |
| 11 | - | 0 |
| 12 | - | 0 |
| 13 | - | 0 |
| 14 | - | 0 |
| 15 | - | 0 |

# Ejercicios (cont.)

b) Suponiendo el algoritmo óptimo para el reemplazo de páginas, mostrar el contenido de la tabla de páginas, tras generar cada una de las siguientes direcciones lógicas: 1112, 1645, 2049, 622, 2776

- Tamaño página: 512 posiciones = $2^9 \Rightarrow$ 9 bits para el offset
- Necesitamos conocer la secuencia de solicitudes de página para aplicar el algoritmo óptimo:

| DL (decimal) | DL (binario) | P | D |
|---|---|---|---|
| 1112 | 10 \| 001011000 | 2 | 88 |
| 1645 | 11 \| 001101101 | 3 | 109 |
| 2049 | 100 \| 000000001 | 4 | 1 |
| 0622 | 1 \| 001101110 | 1 | 110 |
| 2776 | 101 \| 011011000 | 5 | 216 |

# Ejercicios (cont.)

- DL: 1112 = 10 | 001011000 (P = 1112/512= 2)
  - La página 2 no está en memoria $\Rightarrow$ Fallo de TP
  - Como la memoria física está llena, hay que elegir una víctima: 4, 9, 5 ó 1.
  - La que estará más tiempo a ser referenciada es la 9 $\Rightarrow$ sustituimos la 9 por la 2 y actualizamos la TP

| Nº marco | Nº pág. |
|---|---|
| 0 | 4 |
| 1 | 2 |
| 2 | 5 |
| 3 | 1 |

Tabla de páginas

| | Marco | Bit de validez |
|---|---|---|
| 0 | - | 0 |
| 1 | 3 | 1 |
| 2 | 1 | 1 |
| 3 | - | 0 |
| 4 | 0 | 1 |
| 5 | 2 | 1 |
| 6 | - | 0 |
| 7 | - | 0 |

Tabla de páginas

| | Marco | Bit de validez |
|---|---|---|
| 8 | - | 0 |
| 9 | 1 | 0 |
| 10 | - | 0 |
| 11 | - | 0 |
| 12 | - | 0 |
| 13 | - | 0 |
| 14 | - | 0 |
| 15 | - | 0 |

# Ejercicios (cont.)

- DL: 1645 = 11 | 001101101 (P = 1645/512 = 3)
  - La página 3 no está en memoria ⇒ Fallo de TP
  - Como la memoria física está llena, hay que elegir una víctima: 4, 2, 5 ó 1.
  - La que estará más tiempo a ser referenciada es la 2 ⇒ sustituimos la 2 por la 3 y actualizamos la TP

| Nº marco | Nº pág. |
|---|---|
| 0 | 4 |
| 1 | 3 |
| 2 | 5 |
| 3 | 1 |

Tabla de páginas

| | Marco | Bit de validez |
|---|---|---|
| 0 | - | 0 |
| 1 | 3 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |
| 4 | 0 | 1 |
| 5 | 2 | 1 |
| 6 | - | 0 |
| 7 | - | 0 |

Tabla de páginas

| | Marco | Bit de validez |
|---|---|---|
| 8 | - | 0 |
| 9 | 1 | 0 |
| 10 | - | 0 |
| 11 | - | 0 |
| 12 | - | 0 |
| 13 | - | 0 |
| 14 | - | 0 |
| 15 | - | 0 |

  - Para las siguientes peticiones de páginas: 4, 1, 5, no se producen fallos de página porque ya están cargadas

# Ejercicios (cont.)

| | | | | | 1) | 2) | 3) | 4) | 5) |
|---|---|---|---|---|---|---|---|---|---|
| Pg | 4 | 9 | 5 | 1 | 2 | 3 | 4 | 1 | 5 |
| t0 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| t1 | | 9 | 9 | 9 | 2 | 3 | 3 | 3 | 3 |
| t2 | | | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| t3 | | | | 1 | 1 | 1 | 1 | 1 | 1 |

Reemplazo 9

Reemplazo 2

# Ejercicios (cont.)

c) Obtener las direcciones físicas correspondientes a las siguientes direcciones lógicas: 1628, 851, 2700 y 2432

En binario:

- 1628 = 11 | 001011100
  - P 3 ⇒ M 1
  - 1 | 001011100 = 604
- 851 = 1 | 101010011
  - P 1 ⇒ M 3
  - 11 | 101010011 = 1875
- 2700 = 101 | 010001100
  - P 5 ⇒ M 2
  - 10 | 010001100 = 1164
- 2432 = 100 | 110000000
  - P 4 ⇒ M 0
  - 0 | 110000000 = 384

En decimal:

- 1628 = (3, 92)
  - P 3 ⇒ M 1
  - 1 * 512 + 92 = 604
- 851 = (1, 339)
  - P 1 ⇒ M 3
  - 3 * 512 + 339 = 1875
- 2700 = (5, 140)
  - P 5 ⇒ M 2
  - 2 * 512 + 140 = 1164
- 2432 = (4, 384)
  - P 4 ⇒ M 0
  - 0 * 512 + 384 = 384

# Ambito de reemplazo

- Reemplazo Local
  - Selecciona la víctima sólo **entre las páginas residentes del proceso** que ha generado el fallo de página
- Reemplazo Global
  - **Todas las páginas en memoria principal** son candidatas para el reemplazo, independientemente de a qué proceso pertenezcan

# Tamaño del conjunto residente

- **Asignación fija**
  - Proporciona un nº fijo de marcos de memoria principal disponibles para ejecución.
  - Se decide en el instante de creación del proceso
  - Cuando se produzca un fallo de página del proceso en ejecución, la página que se necesite reemplazará una de las páginas del proceso
- **Asignación variable**
  - El nº de páginas asignadas a un proceso varía durante la vida útil del proceso
  - A un proceso que cause una tasa de fallos de página alta de forma continua, se le otorgarán marcos de página adicionales para reducir esa tasa
  - A un proceso con una tasa de fallos excepcionalmente baja (comportamiento ajustado al principio de proximidad) se le reducirán los marcos reservados

# Ambito de reemplazo / tamaño conjunto residente

- Asignación fija, ámbito local
- Asignación fija, ámbito global⇒ no es posible
- Asignación variable, ámbito global
- Asignación variable, ámbito local

# Asignación fija, ámbito local

- Se decide de antemano una cantidad de espacio reservado para un proceso (nº de marcos fijo)
- Si la asignación es demasiado pequeña, habrá una tasa de fallos de página alta
- Si la asignación es demasiado grande, habrá muy pocos programas en la memoria principal $\Rightarrow$ procesador ocioso
- Se aplican los algoritmos de selección de víctima básicos

# Asignación variable, ámbito global

- Combinación más fácil de implementar
- Adoptado por muchos sistemas operativos
- Cada proceso en memoria principal tiene un nº de marcos asignados
- El SO mantiene una lista de **marcos libres**
- Cuando se produce un fallo de página, se añade un marco libre al conjunto residente del proceso y se trae la página a dicho marco
- Si no hay un marco libre, se elige como víctima a una página de entre todos los marcos (puede pertenecer a cualquiera de los procesos residentes)
- Se aplica ***buffering* de páginas**, para evitar el problema con las páginas modificadas que han de ser desalojadas

# *Buffering* de páginas

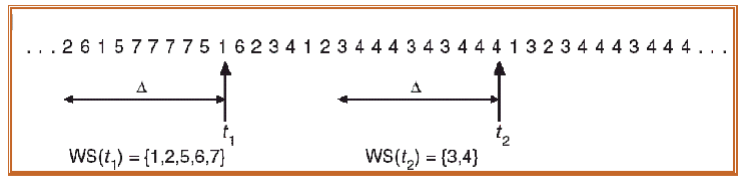
- Mantiene una reserva de **marcos libres**. Cuando se produce un fallo de página, siempre se usa un marco libre (es decir, en verdad no hay reemplazo)
- Cuando el número de marcos libres queda por debajo de cierto umbral se activa un "**demonio de paginación**", que aplica repetidamente el algoritmo 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)

# Asignación variable, ámbito local

- Cuando se carga un nuevo proceso en memoria principal, se le asigna un nº de marcos de página a su conjunto residente, basado en el tipo de aplicación, solicitudes del programa, u otros criterios
- Cuando se produce fallo de página, la víctima pertenecerá al conjunto residente del proceso que causó el fallo
- De vez en cuando se reevalúa la asignación proporcionada a cada proceso
- Se aplican las estrategias de
  - **conjunto de trabajo**
  - **frecuencia de fallos de página**
  - **conjunto de trabajo con muestreos en intervalos variables**

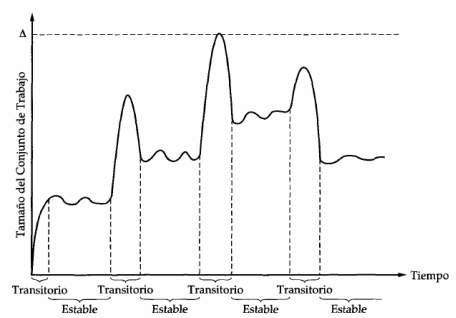
# Conjunto de trabajo o área activa

- Concepto que trata de aproximarse a la localidad actual de un proceso
- Es el conjunto de páginas con el que ha trabajado un proceso en un pasado reciente (ventana del área activa) :
  - WS(t,Δ) = páginas accedidas entre t y t - Δ
- Si tiene el tamaño adecuado, es una fiel aproximación de la localidad actual




# Conjunto de trabajo o área activa

- Generalmente los programas alternan **periodos estables** del **tamaño de su conjunto de trabajo** (debido al principio de proximidad de referencia) con **periodos transitorios** (cambio del programa a una nueva región de referencia)



# Conjunto de trabajo o área activa

- Durante la fase de transición algunas páginas del antiguo conjunto de referencia permanecerán dentro de la **ventana Δ** causando un rápido incremento del tamaño del conjunto de trabajo a medida que se van referenciando nuevas páginas
- A medida que la ventana se desplaza de estas referencias, el tamaño del conj. de trabajo se reduce hasta que contiene únicamente aquellas páginas de la nueva región de referencia

# Conjunto de trabajo o área activa

- Si nº marcos disponibles < tamaño del cjto de trabajo: fallos de pág. frecuentes
- Peligro: hiperpaginación
  - proceso pasa más tiempo intercambiando págs. que ejecutándose, y puede 'robar' págs. de otros procesos, provocando su hiperpaginación (*trashing*)
- Problema:
  - ¿**Cómo calcular nº de marcos** que un proceso necesita?
  - ¿**Cómo calcular tamaño del conjunto de trabajo** para un proceso?

# Conjunto de trabajo o área activa

- Estrategia:
  - El SO vigila el área activa de cada proceso y le asigna un número de marcos igual a su tamaño
    - El área activa se puede estimar utilizando el bit a (historial de uso)
  - Se eliminan periódicamente del conjunto residente aquellas páginas que no se encuentran en el conjunto de trabajo (~LRU)
  - Un proceso se puede ejecutar sólo si su conjunto de trabajo se encuentra en la memoria principal

# Conjunto de trabajo o área activa

- **Ejercicio**:

  **E**n un sistema de gestión de memoria virtual se decide utilizar un modelo de área activa para controlar la demanda de memoria.

  En el sistema se ejecutan actualmente 3 procesos A, B y C. Cada acceso se codifica de forma que el primer símbolo representa el proceso que lo realiza y el segundo la página accedida (se supone que todas las páginas accedidas están dentro del espacio lógico del proceso).

  Suponiendo que el tamaño de la ventana de área activa es 4, calcule el mínimo y máximo tamaño de área activa de cada proceso después de la siguiente secuencia de accesos y suponiendo que el muestreo se inicia inmediatamente antes del primer acceso a cada proceso:

  A1, A1, B0, C3, A0, A1, B0, B1, A3, B3, C0, C1, C1, A1, C0, A0, B0, A2, A2, A1, C0, C1, B1, C2, A3, C3, A4, C3, A2, B0, B0, C2, C3, A0, C2, C0, A4, C0, A1, A0, A0, C0.

# Conjunto de trabajo o área activa

- **Solución**:

  Área activa: conjunto de páginas accedidas en las últimas 4 referencias.

  El tamaño del área activa (TAA) del proceso $p_i$ es el número de páginas referenciadas en la ventana más reciente.

  Extraemos ordenadamente las referencias a páginas para cada proceso y las analizamos en intervalos de 4 referencias.

  Referencias al proceso A:

  A1, A1, A0, A1, A3, A1, A0, A2, A2, A1, A3, A4, A2, A0, A4, A1, A0, A0.

  Se observa que para el proceso A en esta secuencia de referencias el número mínimo de páginas distintas que se referencia tomando una ventana de 4 referencias es 2 páginas y el máximo es de 4 páginas distintas.

  Por tanto tendremos:

  $maxTAA_A = 4$; $minTAA_A = 2$

# Conjunto de trabajo o área activa

- **Solución (cont.)**:

  Referencias al proceso B:

  B0, B0, B1,B3, B0, B1,B0, B0

  Se observa que para el proceso B en esta secuencia de referencias el número mínimo de páginas distintas que se referencia tomando una ventana de 4 referencias es 2 páginas y el máximo es de 3 páginas distintas.

  Por tanto tendremos:

  $maxTAA_B = 3$; $minTAA_B = 2$

# Conjunto de trabajo o área activa

- **Solución (cont.)**:

  Referencias al proceso C:

  C3, C0, C1, C1, C0, C0, C1, C2, C3, C3, C2, C3, C2, C0, C0, C0.

  Se observa que para el proceso C en esta secuencia de referencias el número mínimo de páginas distintas que se referencia tomando una ventana de 4 referencias es 2 páginas y el máximo es de 3 páginas distintas.

  Por tanto tendremos:

  $maxTAA_C = 3$; $minTAA_C = 2$

# Conjunto de trabajo o área activa

- **Ejercicio**:

  En un sistema operativo se ha decidido utilizar el modelo del área activa para controlar el número de páginas que deben residir en memoria por cada proceso.

  La estrategia utilizada es la siguiente:

  - Cada vez que se realiza un acceso a memoria por parte de un proceso, su área activa es recalculada.
  - En todo momento sólo deben estar presentes en memoria las páginas que forman las áreas activas de todos los procesos, siempre y cuando exista disponibilidad de memoria física.
  - Si al recalcular el área activa de un proceso P, se detecta que no hay suficiente memoria física, este proceso P es eliminado completamente de memoria y es suspendido.
  - Cuando se produzca nueva disponibilidad de memoria física, debe cargarse en memoria el área activa del primer proceso que se encuentre suspendido

  En este sistema los parámetros que lo caracterizan son los siguientes:

  - El número de referencias que se utilizan para calcular el área activa es 3
  - Existen 8 marcos en memoria física

# Conjunto de trabajo o área activa

- **Ejercicio (cont.)**:
  - Inicialmente la memoria física esta vacía y llegan simultáneamente, pero en el orden indicado, cuatro procesos (A, B, C y D).
  - En la siguiente tabla se indica la secuencia de accesos a **páginas** que realiza cada proceso dentro de su espacio lógico de direcciones. En el último acceso aparece la letra "T", la cual indica que el proceso termina, lo que implica que sus páginas deben ser eliminadas de memoria y debe atenderse la reanudación de un proceso suspendido si fuera necesario.
  - El sistema atiende a los procesos siguiendo un turno rotatorio, de tal forma que permite a un proceso emitir una dirección y luego se lo permite al siguiente. Ejemplo: A0, B8, C3, D0, A0, B3 .....

| | | | | | | |
|---|---|---|---|---|---|---|
| A | 0 | 0 | 0 | 7 | 8 | T |
| B | 8 | 3 | 5 | 1 | 5 | T |
| C | 3 | 4 | 5 | 6 | 7 | T |
| D | 0 | 1 | 2 | 2 | 1 | T |

# Conjunto de trabajo o área activa

- **Ejercicio (cont.)**:
  - Complete la siguiente tabla de tal forma que describa para cada acceso a memoria en que estado queda la memoria física tras dicho acceso, expresando para cada marco el identificativo del proceso y el número de página que alberga.

| Instante | Proceso Página | Marco 0 | Marco 1 | Marco 2 | Marco 3 | Marco 4 | Marco 5 | Marco 6 | Marco 7 |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

# Conjunto de trabajo o área activa

- **Solución**:
  - En el instante 9 y después de hacer referencia a la página 5 del proceso B, la memoria se encuentra totalmente llena.
  - En el instante 10 el área activa del proceso C no puede ser ubicada en memoria y el proceso C es suspendido y libera memoria.

| Instante | Proceso Página | Marco 0 | Marco 1 | Marco 2 | Marco 3 | Marco 4 | Marco 5 | Marco 6 | Marco 7 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | A0 | A0 | | | | | | | |
| 1 | B8 | A0 | B8 | | | | | | |
| 2 | C3 | A0 | B8 | C3 | | | | | |
| 3 | D0 | A0 | B8 | C3 | D0 | | | | |
| 4 | A0 | A0 | B8 | C3 | D0 | | | | |
| 5 | B3 | A0 | B8 | C3 | D0 | B3 | | | |
| 6 | C4 | A0 | B8 | C3 | D0 | B3 | C4 | | |
| 7 | D1 | A0 | B8 | C3 | D0 | B3 | C4 | D1 | |
| 8 | A0 | A0 | B8 | C3 | D0 | B3 | C4 | D1 | |
| 9 | B5 | A0 | B8 | C3 | D0 | B3 | C4 | D1 | B5 |

# Conjunto de trabajo o área activa

- **Solución (cont.)**:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | D2 | A0 | B8 | D2 | D0 | B3 | | D1 | B5 |
| 11 | A7 | A0 | B8 | D2 | D0 | B3 | A7 | D1 | B5 |
| 12 | B1 | A0 | B1 | D2 | D0 | B3 | A7 | D1 | B5 |
| 13 | D2 | A0 | B1 | D2 | | B3 | A7 | D1 | B5 |
| 14 | A8 | A0 | B1 | D2 | A8 | B3 | A7 | D1 | B5 |
| 15 | B5 | A0 | B1 | D2 | A8 | | A7 | D1 | B5 |
| 16 | D1 | A0 | B1 | D2 | A8 | | A7 | D1 | B5 |
| 17 | AT | C3 | B1 | D2 | C4 | | | D1 | B5 |
| 18 | BT | C3 | | D2 | C4 | | | D1 | |
| 19 | C5 | C3 | C5 | D2 | C4 | | | D1 | |
| 20 | DT | C3 | C5 | | C4 | | | | |
| 21 | C6 | C6 | C5 | | C4 | | | | |
| 22 | C7 | C6 | C5 | | C7 | | | | |
| 23 | CT | | | | | | | | |

# Conjunto de trabajo o área activa

- **Solución (cont.)**: área activa de cada proceso en cada instante:

| | Área activa Proceso A | Área activa Proceso B | Área activa Proceso C | Área activa Proceso D |
|---|---|---|---|---|
| Instante 10 | 0 | 8, 3, 5 | 3, 4, 5<br>Se suspende C | 0, 1 |
| Instante 11 | 0, 7 | 8, 3, 5 | Suspendido | 0, 1, 2 |
| Instante 12 | 0, 7 | 3, 5, 1 | Suspendido | 0, 1, 2 |
| Instante 13 | 0, 7 | 3, 5, 1 | Suspendido | 1, 2 |
| Instante 14 | 0, 7, 8 | 3, 5, 1 | Suspendido | 1, 2 |
| Instante 15 | 0, 7, 8 | 5, 1 | Suspendido | 1, 2 |
| Instante 16 | 0, 7, 8 | 5, 1 | Suspendido | 1, 2 |
| Instante 17 | Libera memoria | 5, 1 | Se ubica su área activa | 1, 2 |
| Instante 18 | ----- | Libera memoria | Se ubica su área activa | 1, 2 |
| Instante 19 | ----- | ----- | 3, 4, 5 | 1, 2 |
| Instante 20 | ----- | ----- | 3, 4, 5 | Libera memoria |
| Instante 21 | ----- | ----- | 4, 5, 6 | ----- |
| Instante 22 | ----- | ----- | 5, 6, 7 | ----- |
| Instante 23 | ----- | ----- | Libera memoria | ----- |

# Conjunto de trabajo o área activa

- **Ejercicio:**

  Dada la siguiente secuencia de referencias a páginas:

  24, 15, 18, 23, 24, 17, 18, 24, 18, 17, 17, 15, 24, 17, 24, 18

  determinar el conjunto de trabajo de un proceso definido para cada uno de los siguientes tamaños de ventana: 2, 3,4 y 5

# Conjunto de trabajo o área activa

- **Solución:**

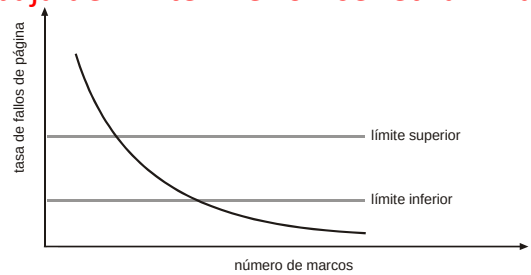
| Referencia a páginas | Tamaño de ventana Δ | | | |
|---|---|---|---|---|
| | 2 | 3 | 4 | 5 |
| 24 | 24 | 24 | 24 | 24 |
| 15 | 24 15 | 24 15 | 24 15 | 24 15 |
| 18 | 15 18 | 24 15 18 | 24 15 18 | 24 15 18 |
| 23 | 18 23 | 15 18 23 | 24 15 18 23 | 24 15 18 23 |
| 24 | 23 24 | 18 23 24 | . | . |
| 17 | 24 17 | 23 24 17 | 18 23 24 17 | 15 18 23 24 17 |
| 18 | 17 18 | 24 17 18 | . | 18 23 24 17 |
| 24 | 18 24 | . | 24 17 18 | . |
| 18 | . | 18 24 | . | 24 17 18 |
| 17 | 18 17 | 24 18 17 | . | . |
| 17 | 17 | 18 17 | . | . |
| 15 | 17 15 | 17 15 | 18 17 15 | 24 18 17 15 |
| 24 | 15 24 | 17 15 24 | 17 15 24 | . |
| 17 | 24 17 | . | . | 17 15 24 |
| 24 | . | 24 17 | . | . |
| 18 | 24 18 | 17 24 18 | 17 24 18 | 15 17 24 18 |

# Algoritmo de frecuencia de fallos de página (PFF)

- En lugar de determinar qué páginas forman parte de conjunto de trabajo, determinemos al menos cuántas son
- Asignemos dinámicamente el nº suficiente de marcos
- Un buen criterio de sustitución nunca debería quitar una página del conjunto de trabajo

# Algoritmo de frecuencia de fallos de página (PFF)

- Para ajustar número de marcos: se mide frecuencia de fallos de página
  - Se fija límite superior e inferior para dicha frecuencia
  - Si se supera límite superior: se asignan más marcos
  - Si se baja de límite inferior: se retiran marcos




# Algoritmo de frecuencia de fallos de página (PFF)

- El algoritmo PFF no funciona bien en los periodos de transición, cuando el proceso cambia el conjunto de páginas que utiliza
  - Ninguna página sale del conjunto residente antes de que hayan pasado F unidades de tiempo virtual desde su última referencia
  - El conjunto residente crece mucho porque hay muchos fallos de página, antes de que las páginas de la antigua región se expulsen

# Política VSWS (cjto. de trabajo con muestreos en intervalos variables)

- En vez de F se definen los siguientes parámetros:
  - M: Duración mínima del intervalo de muestreo.
  - L: Duración máxima del intervalo de muestreo.
  - Q: Número de fallos de página permitidos entre cada par de muestras.
- El algoritmo VSWS es el siguiente:
  - Inicialmente Q = 0 y todos los bits a = 0
  1. Si el tiempo virtual del intervalo llega a L, el intervalo termina y se exploran los bits de acceso
  2. Si se producen Q fallos de página en el intervalo
     a. Si el tiempo virtual del intervalo es menor que M, continuar con el intervalo y se exploran los bits de acceso
     b. Si el tiempo virtual del intervalo es mayor que M, el intervalo termina y se exploran los bits de acceso

# Política VSWS

- Intenta reducir el pico de solicitudes de memoria causadas por una transición abrupta entre 2 áreas de referencia incrementando la frecuencia de muestreo
- Durante el intervalo, las páginas siempre se añaden al conjunto residente. Eso hace que aumente de tamaño
- El conjunto residente sólo puede disminuir de tamaño al final del intervalo
- Cuando hay muchos fallos de página, el intervalo se reduce y el conjunto residente se vacía con mayor frecuencia

- Este algoritmo se utiliza cuando la caché se agota y me quedo sin memoria