Tema 3 - Gestión de memoria

- ▼ Jerarquía de memoria (en orden de mayor a menor rapidez)
  - ▼ CPU
    - Registros procesador → mayor velocidad de lectura
    - ▼ Caché del procesador
      - ▼ Principio de localidad
        - Si un dato/instrucción es accedido es altamente probable que en un tiempo cercano vuelva a ser accedido o lo sean datos/instrucciones cercanos
      - ▼ Acierto de caché
        - Se encuentra un item concreto que se estaba buscando
        - Se evita acceso a espacio de almacenamiento más lento
        - Caso contrario → fallo de caché
      - Por principio de localidad → muchos más aciertos de caché que fallos
  - Memoria principal (RAM)
  - Memoria auxiliar → requiere acceso a bus E/S
- ▼ Espacio físico y lógico
  - ▼ Características
    - Espacio lógico → direcciones lógicas generadas por un programa
    - Espacio físico → direcciones físicas correspondientes a direcciones lógicas en un momento concreto
    - Dirección física → BASE (primera línea de ejecución) + Dirección lógica
    - MMU → dispositivo hardware que traduce direcciones lógicas en direcciones físicas
  - ▼ Ejemplo
    - Si un proceso se carga en una misma zona de memoria que su cabecera, el direccionamiento físico no altera la ejecución del proceso
    - Si un proceso se carga en otra zona de memoria, el direccionamiento físico es diferente y cuando intenta leer JNZ 12 apunta a una sentencia inexistente

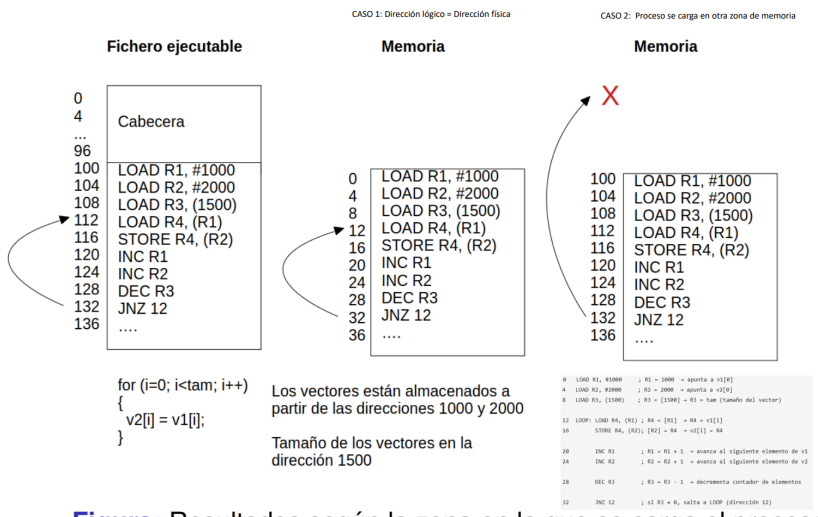


Figura: Resultados según la zona en la que se carga el proceso

- ▼ Asignación de memoria

▼ Contigua

- Asignar todo el almacenamiento necesario para un proceso en un único bloque de posiciones contiguas de memoria
- Genera una alta fragmentación externa

▼ No contigua

- Divide el proceso en bloques que se almacenan en zonas no necesariamente contiguas

▼ Métodos

▼ Segmentación

- Consiste en la aplicación de asignación contigua por regiones
- Produce fragmentación externa

▼ Paginación

▼ Características

- Memoria principal dividida en bloques → marcos de página
- Mapa del proceso dividido en páginas
- Marcos y páginas de igual tamaño
- Para disponer de un contenido en MP → se almacena la página que lo contiene en un marco de página
- Minimiza fragmentación externa

▼ Tablas de páginas

- MMU realiza traducción de página al marco de la página en la que se encuentra mediante una tabla de páginas

▼ Crecimiento de espacio de almacenamiento

- Implementación de esquemas multinivel para evitar tablas de páginas demasiado grandes
- Sólo se almacena en memoria la parte de la tabla de páginas que es requerida en cada momento

- Tamaño de página

Tamaño de página	Fragmentación interna	Coste de gestión
Aumenta	Aumenta	Disminuye
Disminuye	Disminuye	Aumenta

▼ Paginación multinivel

- Implementación de esquema multinivel para evitar tablas de páginas demasiado grandes
- Paginación de la propia tabla de páginas
- Direcciones lógicas → tablas de páginas de cada nivel que hay que recorrer para obtener el marco de página

▼ Ejemplos

▼ Paginación

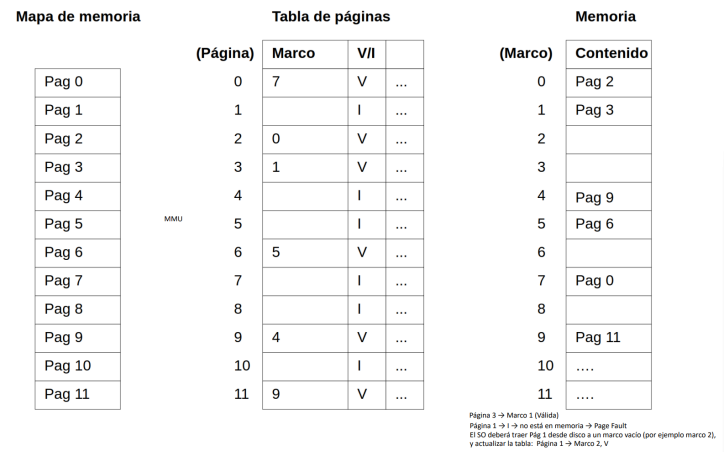
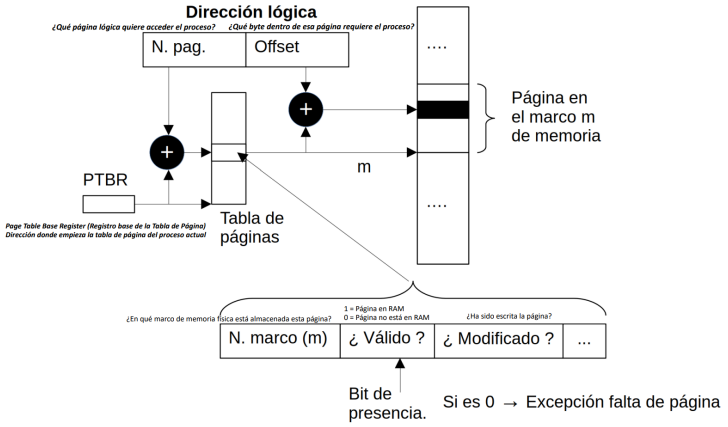


Figura: Ejemplo de paginación

▼ Traducción de direcciones lógicas mediante paginación



▼ Paginación multinivel (2 niveles)

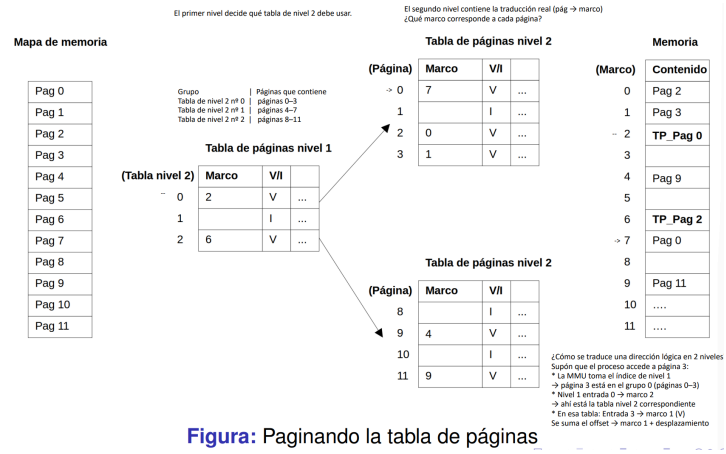


Figura: Paginando la tabla de páginas

▼ Memoria virtual

▼ Características

- Ejecuta procesos cuyo mapa de memoria es mayor que la memoria principal disponible
- Aumenta el nivel de multiprogramación → ejecuta más procesos de los que caben en memoria principal

▼ Utiliza parte del almacenamiento secundario

- Almacena contenido de memoria principal
- Segura que en memoria principal estén elementos que son necesarios para cada instante

- ▼ Paginación y faltas de página
  - Páginas que se requieren pueden no estar en memoria principal → falta de página
  - ▼ Implicaciones
    - Bloquear proceso → otro proceso recibe uso de procesador
    - Encontrar página solicitada en memoria secundaria
    - Encontrar marco libre / desplazar una página de memoria principal a memoria secundaria
    - Cargar página desde disco al marco asignado
    - Actualizar tabla de páginas (bit de presencia, permisos, etc.)
    - Desbloquear proceso → reiniciar instrucción que originó falta de página

▼ Algoritmos de reemplazo y reparto de espacio

- ▼ Características
  - Sustituye la página que va a tardar más en referenciarse
  - Algoritmo no implementable → no es posible saber el patrón de acceso a páginas
- ▼ Algoritmos
  - ▼ FIFO
    - Sustituye página más antigua
    - Presenta problemas con páginas utilizadas durante largos intervalos de tiempo
  - ▼ LRU (Last Recently Used)
    - Sustituye la página que fue objeto de la referencia más antigua
  - ▼ Reloj
    - ▼ Características
      - Versión mejorada de FIFO → proporciona segunda oportunidad a páginas más usadas
      - Evita que páginas muy utilizadas sean eliminadas por llevar mucho tiempo en memoria
      - Cada página tiene asociada un bit de referencia → cuando se usa, ese bit = 1
      - ▼ Si a una página le corresponde ser sustituida (por edad) y tiene bit = 1
        - bit = 0
        - Se pasa a siguiente página candidata
      - Lista circular donde se utiliza un puntero para señalar la posición candidata a ser ocupada por la siguiente página que requiera ser cargada

- X Falta de página con todos los marcos de página en uso
- * Bit de referencia con valor 1. Su valor es 0 en otro caso

▼ Basado en WS (conjunto de trabajo)

▼ Características

- Generalmente los procesos disponen en memoria de un conjunto de páginas constante durante un intervalo de tiempo → conjunto de trabajo
- Para conseguir un buen rendimiento es necesario mantener en memoria principal las páginas del conjunto de trabajo de los procesos que se reparten el tiempo de CPU

▼ Se intenta evitar hiperpaginación

▼ A partir de cierto nivel de multiprogramación

- Coste de faltas de página domina sobre el resto
- Rendimiento del sistema cae en picado

▼ Ejemplo

- Si se cumplen X instantes sin usar una página, se deja de mantener en memoria
- Falta de memoria (*) → la página requerida no está cargada previamente en memoria

- Proceso de 5 páginas. Se muestran las cargadas en memoria principal en cada instante.
- $X = 4$ Algoritmo WS mantiene en memoria solo las páginas que se han usado en los últimos 4 instantes.
- El un instante t, el conjunto de trabajo se considera formado por las páginas referenciadas en el intervalo (t-X,t]. Estas son las que se mantienen cargadas en memoria.

P. requerida	E	D	A	C	C	D	B	C	E	C	E	A	D
	-	-	A	A	A	A	-	-	-	-	-	A	A
	-	-	-	-	-	-	B	B	B	B	-	-	-
	-	-	-	C	C	C	C	C	C	C	C	C	C
	-	D	D	D	D	D	D	D	D	-	-	-	D
	E	E	E	E	-	-	-	-	E	E	E	E	E
¿Falta de p.?	*	*	*	*			*		*			*	*
	{1-4, 1] {1}		{3-4, 3] {1,2,3}			{6-4, 6] {3,4,5,6}							

- * Falta de página

▼ FFP (frecuencia de falta de página)

- Cuando se obtenga un n (tiempo sin falta de página) > Y se eliminan todas las páginas que no se hayan utilizado/leído desde la última vez que hubo "error de lectura de página" (?) → último momento con asterisco
- Cuando hay X (falta de página) → también se añade *

- Proceso de 5 páginas. Se muestran las cargadas en memoria principal en cada instante.
- $Y = 2$

-> E no está → falta (*). Tiempo desde la falta anterior = 1 ≤ Y. R = { A }
-- C ya está en memoria → SIN falta. R no cambia.

@ B no está → falta de página. Tiempo desde la falta anterior: falta anterior en acceso 4, ahora estamos en 7 → 7–4 = 3 > Y
→ se marca con X en la tabla. Desde el acceso 4 hasta el 6 se usaron: C, C, D → conjunto { C, D }. Expulsamos las páginas que NO están en {C, D} → se van A y E.
Añadimos la página pedida B. R = { B, C, D }.

P. requerida	A	E	D	C	C	D	B	C	E	C	E	A	D
	A	A	A	A	A	A	-	-	-	-	-	A	A
	-	-	-	-	-	-	B	B	B	B	B	-	-
	-	-	-	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	E
¿Falta de p.?	*	*	*	*			X		*			X	*
	->				--		@						

- * El tiempo respecto a la falta de página anterior **no** es mayor que Y. Se añade la página requerida.
- X El tiempo respecto a la falta de página anterior es mayor que Y. Las páginas no referenciadas desde la falta de página anterior son retiradas.

▼ Gestión de memoria en Linux

- Kernel utiliza paginación multinivel
- ▼ Paginación por demanda y variable
  - Después de reservar memoria se produce una falta de página → se añade bajo demanda
  - La cantidad de páginas en memoria no es fija
- ▼ Política de sustitución global
  - ▼ Si es necesario eliminar páginas de memoria
    - Se seleccionan globalmente
    - Estimación de su utilidad en el futuro
- ▼ Gestión de páginas
  - ▼ Tipos de páginas
    - ▼ Páginas asociadas a ficheros
      - Caché de información almacenada en disco
      - Mayor probabilidad de ser desalojadas de memoria si es necesario liberar espacio
      - Se eligen primero las no modificadas desde su carga
    - ▼ Páginas anónimas
      - No están asociadas a ficheros
      - Resultado de reservas de memoria realizadas dinámicamente → ciclo de vida de las VMA (área de memoria virtual)
    - Ambos se almacenan por separado
  - ▼ Listas
    - ▼ Tipos
      - ▼ Active list
        - Contienen las páginas más usadas → no se consideran candidatas a ser desalojadas
      - ▼ Inactive list
        - Contienen las páginas raramente usadas → candidatas a ser desalojadas

- Páginas más candidatas a ser eliminadas → inactive list de páginas asociadas a ficheros

▼ Páginas no eliminables

- Páginas generadas como resultado de la función mlock() → permite a procesos reservar memoria que nunca será desplazada

▼ Relación correcta entre la longitud de ambas listas

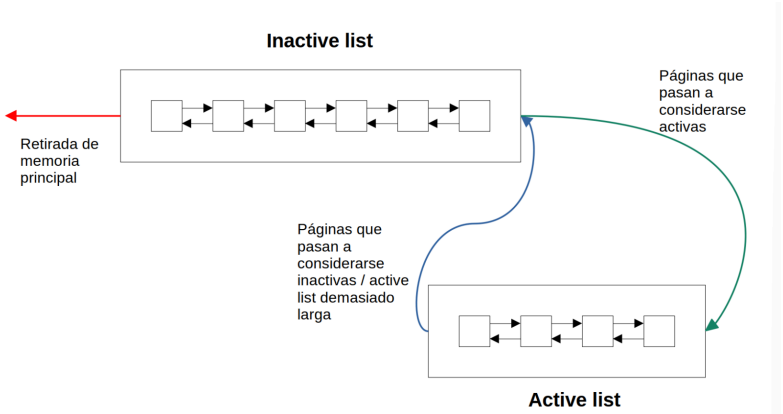
- Si una lista crece demasiado parte de su contenido se traspasa a la otra

▼ Gestión de listas

- Distinta según el tipo
- Depende del kernel
- Páginas de caché de páginas → insertadas en su inactive list correspondiente

▼ Páginas anónimas recién creadas → posibles estrategias

- Insertar en cabecera de inactive list y esperar a que promocionen a active list según su uso
- Insertar directamenteen cabecera de active list



▼ Estimación de actividad

- Nivel hardware → bit que indica si una página ha sido referenciada
- Bit activado por parte de la MMU al acceder a una página
- Bit = 0 con cierta periodicidad
- Si una página es referenciada con bit = 1 → página usada con frecuencia
  - Si la página está en inactive list → pasa a active list
- Uso de recursos hardware generalmente disponibles → no usa termporizadores

▼ Ejercicios

▼ Considere un sistema con un espacio lógico de memoria de 128K páginas (máximo espacio de memoria virtual) con 8 KB cada una, una memoria física de 64 MB y direccionamiento al nivel de byte. ¿Cuántos bits hay en la dirección lógica? ¿Y en la física?

▼ Espacio en dirección lógica

▼ 128K páginas * 8KB cada una = 128.000 * 8 * 1024 = 1.048.576.000 bytes

- 1.048.576.000 direcciones distintas
- Hay que encontrar el mínimo 2^n ≥ 1.048.576.000

- 128 = 2^7
- 8 = 2^3

▼ 1024 = 2^10

        - Además del propio 1024 de pasar los KB a B está el 128 * 1000
        - Como hay que buscar el n mínimo que cubra todas las direcciones hay que buscar el mínimo n tal que 2^n ≥ 1000 → 2^10 = 1024 > 1000
    - 2^10 * 2^10 * 2^7 + * 2^3 = 2^30 → 30 bits de dirección lógica
- ▼ Espacio en dirección física
    - 64 MB = 64 * 1024 * 1024 = 2^6 * 2^10 * 2^10 = 2^26 → 26 bits de dirección física

▼ Considérese un sistema con memoria virtual en el que el procesador tiene una tasa de utilización del 15% y el dispositivo de paginación está ocupado el 97% del tiempo.

- ▼ ¿Qué indican estas medidas?
    - ▼ Hiperpaginación
        - Sistema constantemente moviendo páginas (97%)
        - Muy poco tiempo ejecutando instrucciones CPU (15%)
    - ▼ Se ha llegado a un nivel de multipaginación en el que el rendimiento cae en picado
        - Muchos procesos pasan más tiempo bloqueados por paginación que ejecutándose
- ▼ ¿Y si con el mismo porcentaje de uso del procesador el
porcentaje de uso del dispositivo de paginación fuera del 15%?
    - Se descarta hiperpaginación → dispositivo de paginación no tan ocupado
    - ▼ Puede deberse a otros factores
        - Baja carga de procesos → CPU poco exigida
        - Muchas esperas por E/S o sincronización

▼ Sea un sistema de memoria virtual paginada con direcciones lógicas de 32 bits que proporciona un espacio virtual de 2^20 páginas y con una memoria física de 32 MB ¿cuánta memoria requiere en total un proceso que tenga 453KB, incluida su tabla de páginas cuyas entradas son de 32 bits

- ▼ Reparto de los 32 bits de dirección lógica
    - 2^20 páginas → 20 bits para el número de página
    - ▼ 12 bits → offset
        - 2^12 = 4096 bytes = 4KB
- ▼ Número de páginas necesarias
    - Proceso de 453KB → 453 * 1024 = 463872 bytes
    - 463872 / 4096 bytes (offset) = 113,25 → 114 (redondeo superior para completar páginas)
- ▼ Memoria requerida por el proceso
    - 114 páginas * 4096 bytes de información = 466.944 bytes = 456 KB
    - Al paginar el proceso se incrementa el tamaño del mismo ya que hay que guardar información de cada página
- ▼ Memoria requerida por tabla de paginación
    - Nº de entradas = nº de páginas → 2^20 entradas
    - Tamaño de cada entrada = 32 bits / 8 = 4 bytes
    - ▼ Tamaño de tabla
        - 2^20 * 4 = 2^20 * 2^2 = 2^22 = 4.194.304 bytes = 4MB
        - 4MB / 4KB/página (offset de cada dirección lógica) = 4096KB / 4KB/página = 1024 páginas
- ▼ Memoria total del proceso
    - ▼ Por bytes
        - 466.944 bytes de proceso + 4.194.304 bytes de tabla = 4.661.248 bytes

        - 4.661.248 bytes = 4.552 KB
    ▼ Por páginas
        - 114 páginas de proceso + 1024 páginas de tabla = 1138 páginas
        - 1138 páginas * 4096 bytes/página = 4.661.248 bytes
        - 4.661.248 bytes = 4.552 KB

▼ Un ordenador tiene 4 marcos de página. En la siguiente tabla se muestran: el tiempo de carga, el tiempo del último acceso y los bits R y M para cada página (los tiempos están en tics de reloj). Responda a las siguientes cuestiones justificando su respuesta

▼ ¿Qué página se sustituye si se usa el algoritmo FIFO?

- Se sustituye la página 2, ya que es la que tiene el menor tiempo de carga (es decir, es la página más antigua)

▼ ¿Qué página se sustituye si se usa el algoritmo LRU?

- Se sustituye la página 3, ya que es la página que tiene el menor tiempo de última referencia (es decir, es la página menos utilizada frecuentemente)

▼ ¿Depende el tamaño del conjunto de trabajo de un proceso directamente del tamaño del programa ejecutable asociado a él? Justifique su respuesta.

- No. Aunque sí es un factor determinante a la hora de paginar el proceso en memoria, existen otros factores externos que pueden alterar el tamaño del conjunto de trabajo. Por muy grande que sea el proceso, si tiene una buena localidad en memoria y no aparecen en el mismo intervalo de tiempo otros procesos simultáneos con los que repartir la memoria, el tamaño del conjunto de trabajo puede ser menor que el de un proceso más pequeño.

▼ ¿Por qué una cache (o la TLB) que se accede con direcciones virtuales puede producir incoherencias y requiere que el sistema operativo la invalide en cada cambio de contexto y, en cambio, una que se accede con direcciones físicas no lo requiere?

- Porque, precisamente, las direcciones lógicas dependen del contexto de un proceso, mientras que las direcciones físicas. En contextos diferentes, las direcciones lógicas no son las mismas, ya que son directamente dependientes de dicho contexto (lo que provoca que no sean globales y pueda haber incoherencias por entradas obsoletas o que apunten a dos procesos distintos), mientras que las direcciones físicas se mantienen, ya que son direcciones únicas y globales, lo que hace imposible que haya malentendidos como ocurre con las direcciones lógicas.

▼ Un ordenador proporciona un espacio de direccionamiento lógico (virtual) a cada proceso de 65.536 bytes de espacio dividido en páginas de 4096 bytes. Cierto programa tiene un tamaño de región de texto de 32768 bytes, un tamaño de región de datos de 16386 bytes y tamaño de región de pila de 15878. ¿Cabría este programa en el espacio de direcciones? (Una página no puede ser utilizada por regiones distintas). Si no es así, ¿cómo podríamos conseguirlo, dentro del esquema de paginación?

- 65.536 / 4.096 = 16 páginas disponibles
- 32.768 / 4.096 = 8 páginas para texto
- 16.386 / 4.096 = 5 páginas para datos
- 15.878 / 4.096 = 4 páginas para pila
- 8 + 5 + 4 = 17 páginas → no cabe en el espacio proporcionado

▼ Posibles soluciones dentro del esquema de paginación

- No cargar todo el programa directamente → cargarlo por secciones
- ▼ Reducir el tamaño de cada región
    - Concretamente del espacio de datos que necesita un poco más de 4 páginas
- Aumentar el espacio de direccionamiento

▼ Analice qué puede ocurrir en un sistema que usa paginación por demanda si se recompila un programa mientras se está ejecutando. Proponga soluciones a los problemas que pueden surgir en esta situación.

- El archivo ejecutable en disco se actualiza con el nuevo código binario
- El proceso en memoria continúa ejecutando páginas cargadas sin cambios recientes

▼ Problemas

▼ Incoherencias al mezclar código nuevo y antiguo

- Saltos a direcciones inválidas

▼ Posible pérdida de estado al reemplazar páginas

- Páginas modificadas por el proceso pueden perderse o ser sobreescritas

▼ Soluciones

- Evitar recompilación durante ejecución
- Invalidar caché/páginas del proceso en memoria y recargar desde cero
- Usar versiones o instantáneas para actualizar el disco

▼ Para cada uno de los siguientes campos de la tabla de páginas, se debe explicar si es la MMU o el sistema quién los lee y escribe (en este último caso si se activa o desactiva), y en qué momentos

	Quién lo lee	Quién lo escribe
Número de marco	MMU (durante la traducción de dirección virtual a física para acceder a memoria)	SO (cuando se asigna un marco de página físico a una página virtual durante la carga inicial del proceso)
Bit de presencia	MMU (en cada acceso)	SO (activa cuando carga una página, desactiva cuando libera una página)
Bit de protección	MMU (en cada acceso)	SO (al configurar tabla de páginas)
Bit de modificación	SO (periódicamente o al reemplazar páginas)	MMU (en cada escritura → contenido a cambiado)
Bit de referencia	SO (desactiva periódicamente para algoritmos de reemplazo)	MMU (lo activa en cualquier acceso a página)

▼ Suponga que la tabla de páginas para el proceso actual se parece a la de la figura. Todos los números son decimales, la numeración comienza en todos los casos desde cero, y todas las direcciones de memoria son direcciones en bytes. El tamaño de página es de 1024 bytes.

Página virtual	Bit de presencia	Bit de referencia	Bit de modificación	Marco de página
0	0	1	0	4
1	1	1	1	7
2	1	0	0	1
3	1	0	0	2
4	0	0	0	-
5	1	0	1	0

▼ 999

- 999 / 1024 → página 0
- 999 mod 1024 → offset 999

▼ Bit de presencia = 0 → falta de página

- No se puede traducir la dirección física
- No se actualizan bits

▼ 2121

- 2121 / 1024 → página 2
- 2121 mod 1024 → offset 73

▼ Bit de presencia = 1

- Marco de página = 1
- Dirección física → (1 * 1024) + 73 = 1097

- Bit de referencia → se activa (de 0 pasa a 1)

▼ 5400

- 5400 / 1024 → página 5
- 5400 mod 1024 → offset 280

▼ Bit de presencia = 1

- Marco de página = 0
- Dirección física = 280

- Bit de referencia → se activa (de 0 pasa a 1)

▼ Sea la siguiente secuencia de números de página referenciados: 1,2,3,4,1,2,5,1,2,3,4,5 Calcula el número de faltas de página que se producen utilizando el algoritmo FIFO y considerando que el número de marcos de página de que disfruta nuestro proceso es de:

▼ 3 marcos

- Se producen 10 faltas de página

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

▼ 4 marcos

- Se producen 10 faltas de página

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

▼ ¿Se corresponde esto con el comportamiento intuitivo de que disminuirá el número de faltas de página al aumentar el tamaño de memoria de que disfruta el proceso?

- No, ya que depende de qué páginas se reemplacen puede no ser una mejor solución para un caso concreto, aunque en general cuanta más memoria se tenga más probable es que disminuya el número de faltas de página

▼ En la gestión de memoria en un sistema paginado, ¿qué estructura/s de datos necesitará mantener el Sistema Operativo para administrar el espacio libre?

▼ Bitmap de marcos

- 0 → marco libre
- 1 → marco ocupado

▼ Lista enlazada de marcos libres

- Cada nodo representa un nodo libre
- Cada nodo apunta al nodo anterior y al siguiente

▼ ¿Cuánto puede avanzar como máximo la aguja del algoritmo de reemplazo de páginas del reloj durante la selección de una página?

- Una vuelta completa → número total de marcos de página disponibles

▼ Supongamos la siguiente situación: existe un proceso con 7
páginas y tiene asignados 5 marcos de página. Indica el contenido de la memoria después de cada referencia a una página si como algoritmo de sustitución de página utilizamos el LRU (la página no referenciada hace más tiempo). La secuencia de referencias es la indicada en la figura.

- Se producen 8 faltas de página

Referencia	2	1	3	4	1	5
Marcos de página		1	1	1	1	1
	2	2	2	2	2	2
			3	3	3	3
				4	4	4
						5
Falta de página	*	*	*	*		*

▼ ¿Cuál es la ventaja del algoritmo de faltas de página sobre el algoritmo basado en el modelo del conjunto de trabajo utilizando el tamaño de ventana w?

▼ Mejor criterio para liberar memoria

- En vez de simplemente liberar después de n instantes sin tener en cuenta si se están produciendo faltas de página, precisamente evalúa las páginas que están en memoria desde que no se ha necesitado ninguna página nueva

▼ ¿Cuál es la desventaja?

- Mientras haya faltas de página, un proceso puede mantenerse indefinidamente en memoria sin usarse

▼ Supongamos que tenemos un proceso ejecutándose en un sistema paginado, con gestión de memoria basada en el algoritmo de sustitución frecuencia de faltas de página. El proceso tiene 5 páginas (0, 1, 2, 3, 4). Represente el contenido de la memoria real para ese proceso (es decir, indique que páginas tiene cargadas en cada momento) y cuándo se produce una falta de página. Suponga que, inicialmente, está cargada la página 2, el resto de páginas están en memoria secundaria y que no hay restricciones en cuanto al número de marcos de página disponibles. La cadena de referencias a página es: 0 3 1 1 1 3 4 4 2 2 4 0 0 0 0 3 y el parámetro es τ=3

- Al no haber límite de marcos de página disponibles, sólo se produce falta de página cuando se lee por primera vez cada página desde memoria secundaria

Página requerida	0	3	1	1	1	3
	0	0	0	0	0	0
			1	1	1	1
	2	2	2	2	2	2
		3	3	3	3	3
Falta de página	*	*	*			

▼ Describa el funcionamiento del algoritmo de sustitución basado en la frecuencia de faltas de página, con los siguientes datos: 4 marcos de página, en t = 0 la memoria contiene a la página 2. El tamaño de la ventana es τ= 3 y se produce la secuencia de referencias de páginas, 1 4 2 2 2 4 5 5 3 3 5 1 1 1 1 4

Página requerida	1	4	2	2	2	4
	1	1	1	1	1	1
	2	2	2	2	2	2
		4	4	4	4	4

Falta de página	*	*				