**Trabajo Práctico Nº 5**

**Administración de la Memoria Principal**

1.- Explique a que hacen referencia los siguientes términos:

* Dirección Lógica o Virtual
* Dirección Física

***Lógicas***

Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria.

Se debe realizar una traducción a una dirección física.

***Físicas***

La dirección absoluta en la memoria principal.

Una **dirección lógica** es una referencia a una posición de memoria independiente de la asignación actual de datos a memoria; se debe hacer una traducción a dirección física antes de poder realizar un acceso a memoria. Una **dirección relativa** es un caso particular de dirección lógica, en el cual la dirección se expresa como una posición relativa a algún punto conocido, habitualmente el comienzo del programa. Una **dirección física** o dirección absoluta, es una posición real en memoria principal.

2.- En la técnica de Particiones Múltiples, la memoria es divida en varias particiones y los procesos son ubicados en estas, siempre que el tamaño del mismo sea menor o igual que el tamaño de la partición. Al trabajar con particiones se pueden considerar 2 métodos (independientes entre sí):

* Particiones Fijas
* Particiones Dinámicas

a) Explique cómo trabajan estos 2 métodos. Cite diferencias, ventajas y desventajas.

***Partición fija***

En la mayoría de los esquemas de gestión de memoria, se puede suponer que el sistema operativo ocupa una parte fija de memoria principal y que el resto de la memoria está disponible para ser usado por varios procesos. El esquema más sencillo de gestión de la memoria disponible es dividirla en regiones con límites fijos.

Una posibilidad es emplear particiones de igual tamaño. En este caso, cualquier proceso cuyo tamaño sea menor o igual que el tamaño de la partición puede cargarse en cualquier partición libre. Si todas las particiones están ocupadas y no hay procesos residentes en estado Listo o Ejecutando, el sistema operativo puede sacar un proceso de alguna de las particiones y cargar otro proceso de forma que haya trabajo para el procesador.

Las particiones fijas de igual tamaño plantean dos dificultades:

• Un programa puede ser demasiado grande para caber en la partición. En este caso, el programador debe diseñar el programa mediante superposiciones, para que sólo una parte del programa esté en memoria principal en cada instante. Cuando se necesita un módulo que no está presente, el programa de usuario debe cargar dicho módulo en la partición del programa, superponiéndose a los programas y datos que se encuentren en ella.

• El uso de memoria principal es extremadamente ineficiente. Cualquier programa, sin importar lo pequeño que sea, ocupará una partición completa.

Este fenómeno, en el que se malgasta el espacio interno de una partición cuando el bloque de datos cargado sea más pequeño que la partición, se denomina **fragmentación interna.**

***Algoritmo de Ubicación***

Con particiones del mismo tamaño, la ubicación de un proceso en memoria es trivial. Mientras haya alguna partición libre, puede cargarse un proceso en esa partición. Puesto que todas las particiones son de igual tamaño, no importa la partición que se use. Si todas las particiones están ocupadas con procesos que no están listos para ejecutar, uno de esos procesos debe sacarse y hacer sitio para un nuevo proceso. Cuál debe expulsarse es una decisión de planificación.

Con particiones de distintos tamaños, hay dos maneras posibles de asignar los procesos a las particiones. La forma más simple es asignar cada proceso a la partición más pequeña en la que quepa'. En este caso, hace falta una cola de planificación para cada partición, que albergue los procesos expulsados cuyo destinado es dicha partición.

La ventaja de este enfoque es que los procesos están siempre asignados de forma que se minimiza la memoria desperdiciada dentro de cada partición. Sin embargo, aunque esta técnica parece óptima desde el punto de vista de una partición individual, no lo es desde el punto de vista del sistema global. Así pues, una solución mejor sería emplear una única cola para todos los procesos. Cuando se va a cargar un proceso en memoria principal, se selecciona la partición más pequeña disponible que pueda albergar al proceso. Si todas las particiones están ocupadas, se debe tomar una decisión de intercambio.

El uso de particiones de distinto tamaño proporciona cierto grado de flexibilidad a las particiones fijas. Además, ambos tipos de esquema de partición fija son relativamente simples y exigen un software del sistema operativo y una sobrecarga de procesamiento mínimos. Sin embargo, se plantean los problemas siguientes:

• El número de particiones especificadas en el momento de la generación del sistema limita el número de procesos activos (no suspendidos) del sistema.

• Puesto que los tamaños de partición se programan en el momento de la generación del sistema, los trabajos pequeños no hacen un uso eficiente del espacio de las particiones. En un entorno en el que los requisitos básicos de almacenamiento de todos los procesos se conocen de antemano, puede ser una técnica razonable, pero, en la mayoría de los casos, ineficiente.

El uso de la partición fija es casi nulo hoy día.

***Partición Dinámica***

Con la partición dinámica, las particiones son variables en número y longitud. Cuando se trae un proceso a memoria principal, se le asigna exactamente tanta memoria como necesita y no más.

Este método comienza bien, pero, finalmente, desemboca en una situación en la que hay un gran número de huecos pequeños en memoria. Conforme pasa el tiempo, la memoria comienza a estar más fragmentada y su rendimiento decae. Este fenómeno se denomina fragmentación externa y se refiere al hecho de que la memoria externa a todas las particiones se fragmenta cada vez más, a diferencia de la fragmentación interna, que se comentó anteriormente.

Una técnica para superar la fragmentación externa es la compactación: De vez en cuando, el sistema operativo desplaza los procesos para que estén contiguos de forma que toda la memoria libre quede junta en un bloque. La dificultad de la compactación está en que es un procedimiento que consume tiempo, por lo que desperdicia tiempo del procesador.

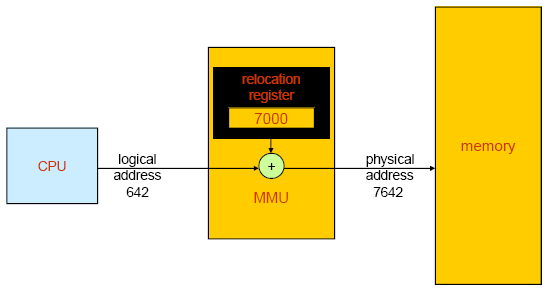
***Algoritmo de Ubicación***

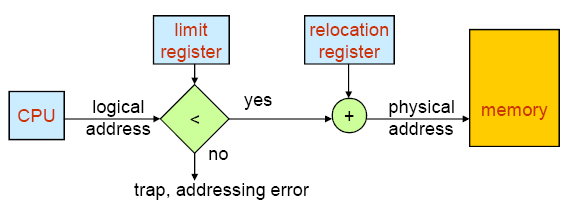
Puesto que la compactación de memoria consume tiempo, atañe al diseñador del sistema operativo decidir adecuadamente cómo asignar un proceso a memoria (como llenar los huecos). Cuando llega el momento de cargar o traer un proceso a memoria principal y, si hay libre más de un bloque de memoria de tamaño suficiente, el sistema operativo debe decidir cuál asignar. Los tres algoritmos de ubicación que se pueden considerar son el del mejor ajuste (best-fit), el del primer ajuste (first-fit) y el del siguiente ajuste (next-fit). Todos ellos se limitan a elegir entre los bloques de memoria libres que son mayores o iguales que el proceso a traer.

b) ¿Qué información debe disponer el SO para poder administrar la memoria con estos métodos?

El sistema operativo mantiene una tabla que indica cuales partes de la memoria están disponibles y cuales están ocupadas.

c) Realice un grafico indicado como realiza el SO la transformación de direcciones lógicas a direcciones físicas.





3.- Al trabajar con particiones fijas, los tamaños de las mismas se pueden considerar:

* Particiones de igual tamaño.
* Particiones de diferente tamaño.

Cite ventajas y desventajas de estos 2 métodos.

Esta contestado en la 2.- a)

4.- Fragmentación

Ambos métodos de particiones presentan el problema de la fragmentación:

* Fragmentación Interna (Para el caso de Particiones Fijas)
* Fragmentación Externa (Para el caso de Particiones Dinámicas)

a) Explique a que hacen referencia estos 2 problemas

Fragmentación

Espacio desperdiciado en la Memoria

Interna 🡺 Particiones Fijas

Espacio dentro de la partición sin utilizar

En este fenómeno se malgasta el espacio interno de una partición cuando el bloque de datos cargado sea más pequeño que la partición.

Externa 🡺 Particiones Dinámicas

El método de la partición dinámica comienza bien, pero, finalmente, desemboca en una situación en la que hay un gran número de huecos pequeños en memoria. Conforme pasa el tiempo, la memoria comienza a estar más fragmentada y su rendimiento decae. Este fenómeno se denomina ***fragmentación externa*** y se refiere al hecho de que la memoria externa a todas las particiones se fragmenta cada vez más, a diferencia de la fragmentación interna, que se comentó anteriormente.

Una técnica para superar la fragmentación externa es la ***compactación***: De vez en cuando, el sistema operativo desplaza los procesos para que estén contiguos de forma que toda la memoria libre quede junta en un bloque. La dificultad de la compactación está en que es un procedimiento que consume tiempo, por lo que desperdicia tiempo del procesador.

Cada vez que entra y sale un proceso se genera huecos en la memoria, en los que eventualmente un proceso no podría entrar, pero si entraría si unimos todos los huecos 🡺 COMPACTACION

b) El problema de la Fragmentación Externa es posible de subsanar. Explique una técnica que evite este problema.

Está contestado arriba. ☺

5.- Paginación

a) Explique cómo trabaja este método de asignación de memoria.

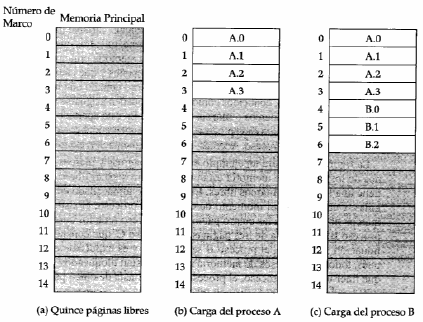
**Paginación Simple**

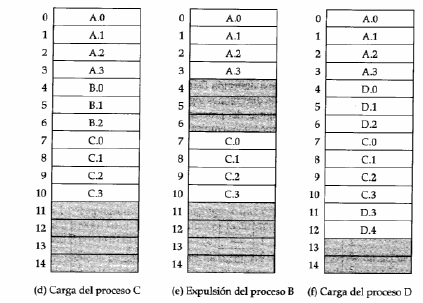
Tanto las particiones de tamaño fijo como las de tamaño variable hacen un uso ineficiente de la memoria; las primeras generan fragmentación interna, mientras que las segundas originan fragmentación externa. Supóngase, no obstante, que la memoria principal se encuentra particionada en trozos iguales de tamaño fijo relativamente pequeños y que cada proceso está dividido también en pequeños trozos de tamaño fijo y del mismo tamaño que los de memoria. En tal caso, los trozos del proceso, conocidos como **páginas,** pueden asignarse a los trozos libres de memoria, conocidos como marcos o **marcos** de página. ***Se verá que el espacio malgastado en memoria para cada proceso por fragmentación interna consta sólo de una fracción de la última página del proceso. Además, no hay fragmentación externa.***

La figura 6.7 muestra un ejemplo del uso de páginas y marcos. En un instante dado, algunos de los marcos de memoria están en uso y otros están libres. ***El sistema operativo mantiene una lista de los marcos libres***. El proceso A, almacenado en disco, consta de cuatro páginas. Cuando llega el momento de cargar este proceso, el sistema operativo busca cuatro marcos libres y carga las cuatro páginas del proceso A en los cuatro marcos (figura 6.7b). El proceso B, que consta de tres páginas y el proceso C, que consta de cuatro, se cargan a continuación. Más tarde, el proceso B se suspende y es expulsado de memoria principal. Después, todos los procesos de memoria principal están bloqueados y el sistema operativo tiene que traer un nuevo proceso, el proceso D, que consta de cinco páginas.

Supóngase ahora, como en este ejemplo, que no hay suficientes marcos sin usar contiguos para albergar al proceso. ¿Impedirá esto al sistema operativo cargar D? La respuesta es negativa, puesto que se puede emplear de nuevo el concepto de dirección lógica. Ya no será suficiente con un simple registro base. ***En su lugar, el sistema operativo mantiene una tabla de páginas para cada proceso***. La tabla de páginas muestra la posición del marco de cada página del proceso. Dentro del programa, cada dirección lógica constará de un número de página y de un desplazamiento dentro de la página. Recuérdese que, en el caso de la partición simple, una dirección lógica era la posición de una palabra relativa al comienzo del programa; el procesador realizaba la traducción a dirección física. Con paginación, el hardware del procesador también realiza la traducción de direcciones lógicas a físicas. Ahora, el procesador debe saber cómo acceder a la tabla de páginas del proceso actual. Dada una dirección lógica (número de página, desplazamiento), el procesador emplea la tabla de páginas para obtener una dirección física (número de marco, desplazamiento).

Continuando con el ejemplo, las cinco páginas del proceso D se cargan en los marcos 4, 5, 6, 11 y 12.





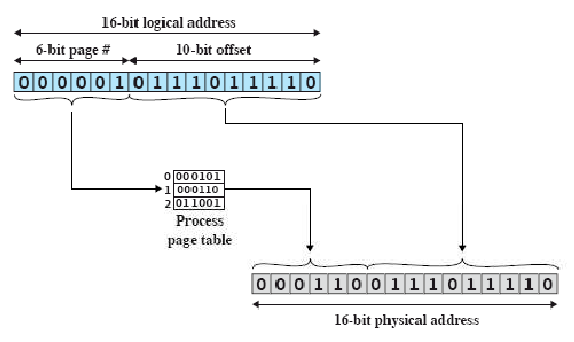
b) ¿Qué estructuras adicionales debe poseer el SO para llevar a cabo su implementación?

c) Explique, utilizando gráficos, como son transformadas las direcciones lógicas en físicas.

d) En este esquema: ¿Se puede producir fragmentación (interna y/o externa)?

Paginación

* La memoria es dividida en pequeños trozos de igual tamaño 🡺 Marcos
* Cada proceso es dividido en trozos de igual tamaño que los marcos 🡺 Paginas
* El SO mantiene una tabla de páginas por cada proceso.
* Contiene el marco en la que está situada cada página.
* La dirección lógica consiste en un número de página y un desplazamiento dentro de la misma.



Para la traducción de direcciones hay que dar los siguientes pasos:

• Obtener el número de página de los n bits más significativos de la dirección lógica.

• Emplear el número de página como índice en la tabla de páginas del proceso para encontrar el número de marco k.

• El comienzo de la dirección física del marco es k x 2^m y la dirección física del byte referenciado es este número más el desplazamiento.

No hace falta calcular esta dirección física, sino que se construye fácilmente concatenando el número de marco con el desplazamiento.

6.- Cite similitudes y diferencias entre la técnica de paginación y la de particiones fijas.

Cada tabla de páginas contiene una entrada por cada página del proceso, por lo que la tabla se indexa fácilmente por número de página (comenzando en la página 0). En cada entrada de la tabla de páginas se encuentra el número de marco en memoria, si lo hay, que alberga la página correspondiente. Además, el sistema operativo mantiene una lista de marcos libres con todos los marcos de memoria que actualmente están vacíos y disponibles para las páginas.

Así pues, se puede comprobar que la paginación simple, tal y como se describe, es similar a la partición estática. Las diferencias están en que, con paginación, las particiones son algo más pequeñas, un programa puede ocupar más de una partición y éstas no tienen por qué estar contiguas.

Resumiendo, mediante la paginación simple, la memoria principal se divide en pequeños marcos del mismo tamaño. Cada proceso se divide en páginas del tamaño del marco; los procesos pequeños necesitaran pocas páginas, mientras que los procesos grandes necesitarán más. Cuando se introduce un proceso en memoria, se cargan todas sus páginas en los marcos libres y se rellena su tabla de páginas. Esta técnica resuelve la mayoría de los problemas inherentes a la partición.

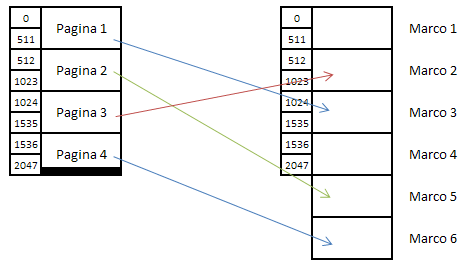
7.- Suponga un sistema donde la memoria es administrada mediante la técnica de paginación, y donde:

* El tamaño de la página es de 512 bytes
* Cada dirección de memoria referencia 1 byte.
* Los marcos en memoria principal se encuentran desde la dirección física 0.

Suponga además un proceso con un tamaño 2000 bytes y con la siguiente tabla de páginas:

|  |  |
| --- | --- |
| Pagina | Marco |
| 0 | 3 |
| 1 | 5 |
| 2 | 2 |
| 3 | 6 |

a) Realice los gráficos necesarios (de la memoria, proceso y tabla de páginas) en el que reflejen el estado descrito.



b) Indicar si las siguientes direcciones lógicas son correctas y en caso afirmativo indicar la dirección física a la que corresponden:

***DIV 🡺marco MOD 🡺desplazamiento***

i) 35

|  |  |  |  |
| --- | --- | --- | --- |
| (35 DIV 512) | + 1 = 1 | (1,35) | 1024 |
| 0 | + 1 = 1 |  | + 35 |
|  |  |  | 1059 |
| (35 MOD 512) | = 35 |  |  |

ii) 512

|  |  |  |  |
| --- | --- | --- | --- |
| (512 DIV 512) | + 1 = 2 | (2,0) | 2048 |
| 1 | + 1 = 2 |  | + 0 |
|  |  |  | 2048 |
| (512 MOD 512) | = 0 |  |  |

iii) 2051

|  |  |  |  |
| --- | --- | --- | --- |
| (2051 DIV 512) | + 1 = 5 | (5,3) 🡺 no existe numero de pagina 5!!!! |  |
| 4 | + 1 = 5 |  |  |
|  |  |  |  |
| (2051 MOD 512) | = 3 |  |  |

iv) 0

|  |  |  |  |
| --- | --- | --- | --- |
| (0 DIV 512) | + 1 = 1 | (1,0) | 1024 |
| 0 | + 1 = 1 |  | + 0 |
|  |  |  | 1024 |
| (0 MOD 512) | = 0 |  |  |

v) 1325

|  |  |  |  |
| --- | --- | --- | --- |
| (1325 DIV 512) | + 1 = 3 | (3,301) | 512 |
| 2 | + 1 = 3 |  | + 301 |
|  |  |  | 813 |
| (1325 MOD 512) | = 301 |  |  |

vi) 602

|  |  |  |  |
| --- | --- | --- | --- |
| (602 DIV 512) | + 1 = 2 | (2,90) | 2048 |
| 1 | + 1 = 2 |  | + 90 |
|  |  |  | 2138 |
| (602 MOD 512) | = 90 |  |  |

c) Indicar, en caso de ser posible, las direcciones lógicas del proceso que se corresponden si las siguientes direcciones físicas:

i) 509

|  |  |  |  |
| --- | --- | --- | --- |
| (509 DIV 512) | + 1 = 1 | (1,509) 🡺 el marco 1 no está vinculado a ninguna pagina!!!! |  |
| 0 | + 1 = 1 |  |  |
|  |  |  |  |
| (509 MOD 512) | = 509 |  |  |

ii) 1500

|  |  |  |  |
| --- | --- | --- | --- |
| (1500 DIV 512) | + 1 = 3 | (3,476) 🡺 0 +476 = 476 🡺 DIRECCION LOGICA |  |
| 2 | + 1 = 3 |  |  |
|  |  |  |  |
| (1500 MOD 512) | = 476 |  |  |

iii) 0

|  |  |  |  |
| --- | --- | --- | --- |
| (0 DIV 512) | + 1 = 1 | (1,0) 🡺 el marco 1 no está vinculado a ninguna pagina!!!! |  |
| 0 | + 1 = 1 |  |  |
|  |  |  |  |
| (0 MOD 512) | = 0 |  |  |

iv) 3215

|  |  |  |  |
| --- | --- | --- | --- |
| (3215 DIV 512) | + 1 = 7 | (7,143) 🡺 el marco 7 no está vinculado a ninguna pagina!!!! |  |
| 6 | + 1 = 7 |  |  |
|  |  |  |  |
| (3215 MOD 512) | = 143 |  |  |

v) 1024

|  |  |  |  |
| --- | --- | --- | --- |
| (1024 DIV 512) | + 1 = 3 | (3,0) 🡺 0 + 0 = 0 🡺 DIRECCION LOGICA |  |
| 2 | + 1 = 3 |  |  |
|  |  |  |  |
| (1024 MOD 512) | = 0 |  |  |

vi) 2000

|  |  |  |  |
| --- | --- | --- | --- |
| (2000 DIV 512) | + 1 = 4 | (4,464) 🡺 el marco 4 no está vinculado a ninguna pagina!!!! |  |
| 3 | + 1 = 4 |  |  |
|  |  |  |  |
| (2000 MOD 512) | = 464 |  |  |

d) ¿Indique, en caso que se produzca, la fragmentación (interna y/o externa)?

Se verá que el espacio malgastado en memoria para cada proceso por fragmentación interna consta sólo de una fracción de la última página del proceso. Además, no hay fragmentación externa.

En el ejemplo la fragmentación (interna) producida es de 48 bytes.

8.- Considere un espacio lógico de 8 páginas de 1024 bytes cada una, mapeadas en una memoria física de 32 marcos.

a) ¿Cuantos bits son necesarios para representar una dirección lógica?

2^3 = 8 🡺 3 bits para las paginas

2^10 = 1024 🡺 10 bits para el desplazamiento

b) ¿Cuantos bits son necesarios para representar una dirección física?

La misma cantidad de bits.

9.- Segmentación

a) Explique cómo trabaja este método de asignación de memoria.

**Segmentación Simple**

Otro modo de subdividir el programa es la segmentación. En este caso, el programa y sus datos asociados se dividen en un conjunto de **segmentos.** No es necesario que todos los segmentos de todos los programas tengan la misma longitud, aunque existe una longitud máxima de segmento. Como en la paginación, una dirección lógica segmentada consta de dos partes, en este caso un número de segmento y un desplazamiento.

Como consecuencia del empleo de segmentos de distinto tamaño, la segmentación resulta similar a la partición dinámica. En ausencia de un esquema de superposición o del uso de memoria virtual, sería necesario cargar en memoria todos los segmentos de un programa para su ejecución. La diferencia, en comparación con la partición dinámica, radica en que, con segmentación, un programa puede ocupar más de una partición y éstas no tienen por qué estar contiguas. La segmentación elimina la fragmentación interna, pero, como la partición dinámica, sufre de fragmentación externa. Sin embargo, debido a que los procesos se dividen en un conjunto de partes más pequeñas, la fragmentación externa será menor.

Mientras que la paginación es transparente al programador, la segmentación es generalmente visible y se proporciona como una comodidad para la organización de los programas y datos. Normalmente, el programador o el compilador asignan los programas y los datos a diferentes segmentos. En aras de la programación modular, el programa o los datos pueden ser divididos de nuevo en diferentes segmentos. El principal inconveniente de este servicio es que el programador debe ser consciente de la limitación de tamaño máximo de los segmentos.

Otra consecuencia del tamaño desigual de los segmentos es que no hay una correspondencia simple entre las direcciones lógicas y las direcciones físicas. De forma análoga a la paginación, ***un esquema de segmentación simple hará uso de una tabla de segmentos para cada proceso y una lista de bloques libres en memoria principal. Cada entrada de tabla de segmentos tendría que contener la dirección de comienzo del segmento correspondiente en memoria principal***. La entrada deberá proporcionar también la longitud del segmento para asegurar que no se usan direcciones no válidas. Cuando un proceso pasa al estado Ejecutando, se carga la dirección de su tabla de segmentos en un registro especial del hardware de gestión de memoria.

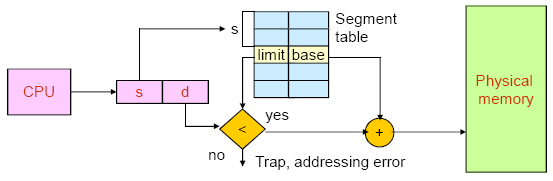
b) ¿Qué estructuras adicionales debe poseer el SO para llevar a cabo su implementación?

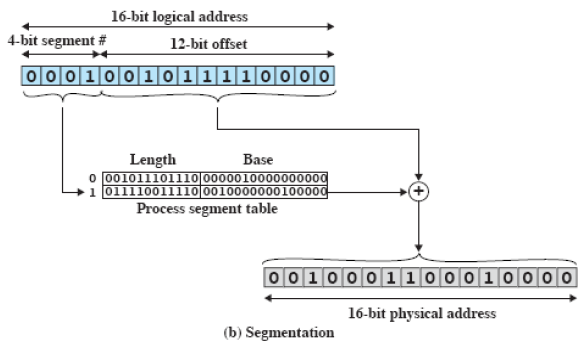
Un esquema de segmentación simple hará uso de una tabla de segmentos para cada proceso y una lista de bloques libres en memoria principal. Cada entrada de tabla de segmentos tendría que contener la dirección de comienzo del segmento correspondiente en memoria principal.

Arquitectura

* Tabla de Segmentos
  + Permite mapear la dirección lógica en física. Cada entrada contiene:
* Base: Dirección física de comienzo del segmento
* Limit: Longitud del Segmento
* Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos.
* Segment-table length register (STLR): cantidad de segmentos de un programa.

c) Explique, utilizando gráficos, como son transformadas las direcciones lógicas en físicas.





d) En este esquema: ¿Se puede producir fragmentación (interna y/o externa)?

La segmentación elimina la fragmentación interna, pero, como la partición dinámica, sufre de fragmentación externa. Sin embargo, debido a que los procesos se dividen en un conjunto de partes más pequeñas, la fragmentación externa será menor.

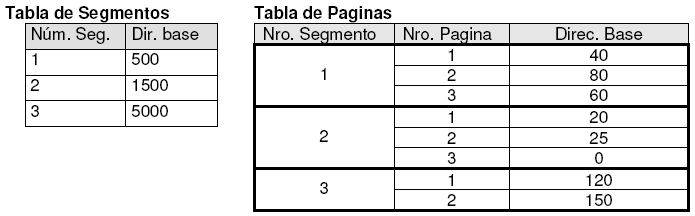
10.- Cite similitudes y diferencias entre la técnica de segmentación y la de particiones dinámicas.

Como consecuencia del empleo de segmentos de distinto tamaño, la segmentación resulta similar a la partición dinámica. En ausencia de un esquema de superposición o del uso de memoria virtual, sería necesario cargar en memoria todos los segmentos de un programa para su ejecución. La diferencia, en comparación con la partición dinámica, radica en que, con segmentación, un programa puede ocupar más de una partición y éstas no tienen por qué estar contiguas.

11.- Cite similitudes y diferencias entre la técnica de paginación y segmentación.

Como en la paginación, una dirección lógica segmentada consta de dos partes, en este caso un número de segmento y un desplazamiento. La segmentación elimina la fragmentación interna, pero, como la partición dinámica, sufre de fragmentación externa. Sin embargo, debido a que los procesos se dividen en un conjunto de partes más pequeñas, la fragmentación externa será menor. Mientras que la paginación es transparente al programador, la segmentación es generalmente visible y se proporciona como una comodidad para la organización de los programas y datos. Normalmente, el programador o el compilador asignan los programas y los datos a diferentes segmentos. En aras de la programación modular, el programa o los datos pueden ser divididos de nuevo en diferentes segmentos. El principal inconveniente de este servicio es que el programador debe ser consciente de la limitación de tamaño máximo de los segmentos. Otra consecuencia del tamaño desigual de los segmentos es que no hay una correspondencia simple entre las direcciones lógicas y las direcciones físicas. De forma análoga a la paginación, un esquema de segmentación simple hará uso de una tabla de segmentos para cada proceso y una lista de bloques libres en memoria principal.

12.- Dado un S.O. que administra la memoria por medio de segmentación paginada, y teniéndose disponibles las siguientes tablas:



Indicar las direcciones físicas correspondientes a las siguientes direcciones lógicas (segmento, página, desplazamiento):

i) (2,1,1)

número de segmento número de pagina desplazamiento

|  |  |
| --- | --- |
| 1500 |  |
| + 20 |  |
| + 1 |  |
| -1 | 🡺 porque empieza de página 1 |
| 1520 |  |

ii) (1,3,15)

|  |  |
| --- | --- |
| 500 |  |
| + 60 |  |
| +15 |  |
| -1 | 🡺 porque empieza de página 1 |
| 574 |  |

iii) (3,1,10)

|  |  |
| --- | --- |
| 5000 |  |
| + 120 |  |
| + 10 |  |
| -1 | 🡺 porque empieza de página 1 |
| 5129 |  |

iv) (2,3,5)

|  |  |
| --- | --- |
| 1500 |  |
| + 0 |  |
| + 5 |  |
| -1 | 🡺 porque empieza de página 1 |
| 1504 |  |

13.- Memoria Virtual

a) Describa que beneficios introduce este esquema de administración de la memoria.

1. Se pueden conservar más procesos en memoria principal. Puesto que se van a cargar sólo algunos fragmentos de un proceso particular, habrá sitio para más procesos. Esto conduce a una utilización más eficiente del procesador, puesto que es más probable que, por lo menos, uno de los numerosos procesos esté en estado Listo en un instante determinado.

2. Es posible que un proceso sea más grande que toda la memoria principal. Se elimina así una de las limitaciones más notorias de la programación. Sin el esquema que se ha expuesto, un programador debe ser consciente de cuánta memoria tiene disponible. Si el programa que está escribiendo es demasiado grande, el programador debe idear formas de estructurar el programa en fragmentos que puedan cargarse de forma separada con algún tipo de estrategia de superposición. Con una memoria virtual basada en paginación o segmentación, este trabajo queda para el sistema operativo y el hardware. En lo que atañe al programador, se las arregla con una memoria enorme, dependiendo del tamaño de almacenamiento en disco. El sistema operativo cargará automáticamente en memoria principal los fragmentos de un proceso cuando los necesita.

La memoria virtual permite una multiprogramación muy efectiva y releva al usuario de las rígidas e innecesarias restricciones de la memoria principal.

Ventajas

* Más procesos pueden ser mantenidos en memoria.
  + Sólo son cargadas algunas partes de cada proceso.
  + Con más número de procesos en la memoria principal es más probable que existan más procesos Ready
* Un proceso puede ser más grande que la memoria Principal
  + El usuario no se debe preocupar por el tamaño de sus programas
  + La limitación la impone el HW y el bus de direcciones.

b) ¿En qué se debe apoyar el SO para su implementación?

El SO debe ser capaz de manejar el movimiento de las páginas (o segmentos) entre la memoria principal y la secundaria.

c) Al implementar está técnica utilizando paginación por demanda, las tablas de páginas de un proceso deben contar con información adicional además del marco donde se encuentra la página. ¿Cuál es está información? ¿Porque es necesaria?

En el estudio de la paginación simple se indicó que cada proceso tiene su propia tabla de páginas y que, cuando carga todas sus páginas en memoria principal, se crea y carga en memoria principal una tabla de páginas. Cada entrada de la tabla de páginas contiene el número de marco de la página correspondiente en memoria principal. Cuando se considera un esquema de memoria virtual basado en la paginación se necesita la misma estructura, una tabla de páginas. Nuevamente, es normal asociar una única tabla de páginas con cada proceso. En este caso, sin embargo, las entradas de la tabla de páginas pasan a ser más complejas. Puesto que sólo algunas de las páginas de un proceso pueden estar en memoria principal, se necesita un bit en cada entrada de la tabla para indicar si la página correspondiente está presente (P) en memoria principal o no lo está. Si el bit indica que la página está en memoria, la entrada incluye también el número de marco para esa página.

Otro bit de control necesario en la entrada de la tabla de páginas es el bit de modificación (M), para indicar si el contenido de la página correspondiente se ha alterado desde que la página se cargó en memoria principal. Si no ha habido cambios, no es necesario escribir la página cuando sea sustituida en el marco que ocupa actualmente. Puede haber también otros bits de control. Por ejemplo, si la protección o la compartición se gestionan a nivel de página, se necesitarán más bits con tal propósito.

14.- Fallos de Página (Page Faults):

a) ¿Cuándo se producen?

En principio, cada referencia a memoria virtual puede generar dos accesos a memoria: uno para obtener la entrada de la tabla de páginas correspondiente y otro para obtener el dato deseado. Así pues, un esquema sencillo de memoria virtual podría tener el efecto de doblar el tiempo de acceso a memoria. Para solucionar este problema, la mayoría de los esquemas de memoria virtual hacen uso de una cache especial para las entradas de la tabla de páginas, llamada generalmente buffer de traducción adelantada (TLB, Translation Lookaside Buffer). Esta cache funciona del mismo modo que una memoria cache y contiene aquellas entradas de la tabla de páginas usadas hace menos tiempo. Dada una dirección virtual, el procesador examinará primero la TLB. Si la entrada de tabla de páginas buscada está presente (un acierto en la TLB), se obtiene el número de marco y se forma la dirección real. Si no se encuentra la entrada de la tabla de páginas buscada (un fallo de TLB), el procesador emplea el número de página para buscar en la tabla de páginas del proceso y examinar la entrada correspondiente de la tabla de páginas. Si se encuentra activo el bit de presencia, es que la página está en memoria principal y el procesador puede obtener el número de marco de la entrada de la tabla de páginas para formar la dirección real. El procesador, además, actualiza la TLB para incluir esta nueva entrada de la tabla de páginas. Por último, si el bit de presencia no está activo, es que la página buscada no está en memoria principal y se produce un fallo en el acceso a memoria, llamado ***fallo de página***. En este punto, se abandona el ámbito del hardware y se invoca al sistema operativo, que carga la página necesaria y actualiza la tabla de páginas.

Fallo de páginas (Page Fault)

* Se intenta acceder a una página que no se encuentra en la memoria principal
  + El bit V es controlado por el HW
* El HW detecta la situación y genera un trap al S.O.
* El S.O. coloca al proceso en estado de “Blocked” (espera)

b) ¿Quién es responsable de detectar un fallo de página?

El hardware.

c) Describa las acciones que emprende el SO cando se produce un fallo de página.

Carga la página necesaria y actualiza la tabla de páginas.

* El S.O. busca un “Frame Libre” y genera una operación de E/S al disco para subir a dicho Frame la porción (página) del proceso que se necesita.
* El SO le asigna la CPU a otro proceso
  + La E/S se realizará y avisará mediante interrupción su finalización.
* Cuando la operación de E/S finaliza el SO:
  + Actualiza la tabla de páginas
    - Coloca el Bit V en 1
    - Coloca la base del Frame donde se colocó la página
  + El proceso que generó el Fallo de Página pasa a estado de Ready (listo)

15.- Direcciones:

a) Si se dispone de un espacio de direcciones virtuales de 32 bits, donde cada dirección referencia 1 byte:

i) ¿Cuál es el tamaño máximo de un proceso (recordar “espacio virtual”)?

Tamaño Máximo de un proceso = 2^32 bytes = 4Gb

2^32 = cantidad de direcciones de 1 byte 🡺 4.294.967.296 bytes = 4.194.304 Kb = 4096 Mb = 4gb

ii) Si el tamaño de pagina es de 512 Kb. ¿Cuál es el número máximo de páginas que puede tener un proceso?

Número Máximo de Paginas por Proceso = 2^32 / 512 = 8388608 🡺 4.194.304 Kb

4.194.304 Kb / 512 = 8192 paginas

iii) Si el tamaño de pagina es de 512Kb. y se disponen de 256 Mb de memoria real ¿Cuál es el número de marcos que puede haber?

Numero de Marcos = 256 \* 1024 / 512 = 512 marcos

256 Mb = 262.164 Kb / 512 Kb = 512 marcos

iv) Si se utilizaran 2 Kb. para cada entrada en la tabla de páginas de un proceso: ¿Cuál sería el tamaño máximo de la tabla de páginas de cada proceso?

Cantidad máximas de páginas = 8192

🡺 8192 páginas \* 2 Kb = 16.384 Kb = 16 Mb es lo que ocuparía la tabla de páginas para ese proceso.

16.- Como se vio en el ejercicio anterior, la tabla de páginas de un proceso puede alcanzar un tamaño considerablemente grande, que incluso, no podría almacenarse de manera completa en la memoria real. Es por esto que el SO también realiza paginación sobre las tablas de páginas.

Existen varios enfoques para administrar las tablas de páginas:

* Tablas de páginas de 1 nivel.
* Tablas de páginas de 2 niveles.
* Tablas de páginas invertidas.

Explique brevemente como trabajan estos enfoques e indique como se realiza la transformación de la dirección virtual en dirección física.

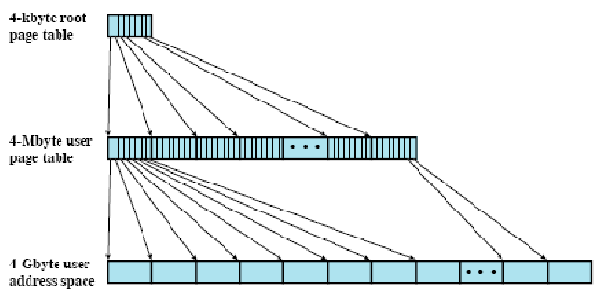
MV con Paginación

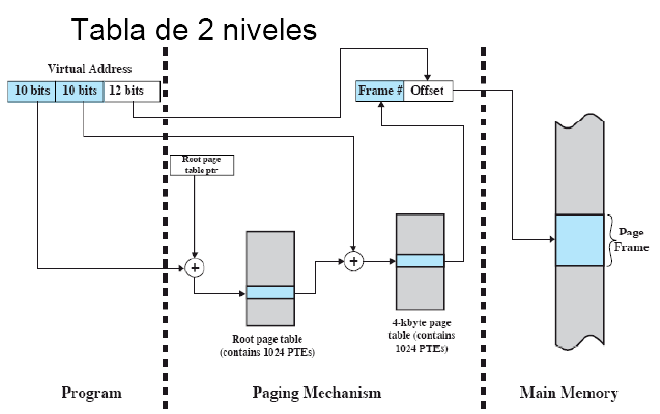
Tabla de Páginas

* Cada proceso tiene su tabla de páginas
* El tamaño de la tabla de páginas depende del espacio de direcciones del proceso.
* Puede alcanzar un tamaño considerable

🡺 El SO pagina las tablas de cada proceso.

* Formas de organizar:
  + Tabla de 1 nivel: Tabla única lineal
  + Tabla de 2 niveles
  + Tabla invertida : Hashing
* La forma de organizarla depende del HW Subyacente





***Estructura de la Tabla de Páginas***

Así pues, el mecanismo básico de lectura de una palabra de memoria supone la traducción por medio de la tabla de páginas de una dirección virtual o lógica, formada por un número de página y un desplazamiento, a una dirección física que está formada por un número de marco y un desplazamiento. Puesto que la tabla de páginas es de longitud variable, en función del tamaño del proceso, no es posible suponer que quepa en los registros. En su lugar, debe estar en memoria principal para ser accesible. La figura 7.3 sugiere una implementación en hardware de este esquema. Cuando se está ejecutando un proceso en particular, la dirección de comienzo de la tabla de páginas para este proceso se mantiene en un registro. El número de página de la dirección virtual se emplea como índice en esta tabla para buscar el número de marco correspondiente. Este se combina con la parte de desplazamiento de la dirección virtual para generar la dirección real deseada.

Considérese el número de entradas necesarias para la tabla de páginas. En la mayoría de los sistemas hay una tabla de páginas por proceso, pero cada proceso puede ocupar una cantidad enorme de memoria virtual.

Evidentemente, la cantidad de memoria dedicada sólo a la tabla de páginas puede ser inaceptable. Para solucionar este problema, la mayoría de los esquemas de memoria virtual almacenan las tablas de páginas en memoria virtual en vez de en memoria real. Esto significa que estas tablas de páginas están también sujetas a paginación, de la misma forma que las otras páginas. Cuando un proceso está ejecutando, al menos una parte de su tabla de páginas debe estar en memoria principal, incluyendo la entrada de la tabla de páginas para la página actualmente en ejecución. Algunos procesadores usan un esquema a dos niveles para organizar grandes tablas de páginas. En este esquema, hay un directorio de páginas en el que cada entrada señala a una tabla de páginas. Así pues, si la longitud del directorio de páginas es *X* y la longitud máxima de una tabla de páginas es *Y,* un proceso puede estar formado por hasta *X* x *Y* páginas. Normalmente, la longitud máxima de una tabla de páginas está limitada a una página.



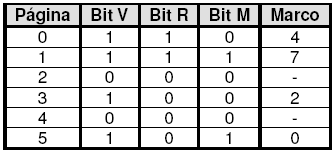
Un enfoque alternativo al uso de tablas de páginas a uno o dos niveles consiste en el uso de una estructura de **tabla de páginas invertida** (figura 7.4).



Con este método, la parte del número de página en una dirección virtual se contrasta en una tabla de dispersión por medio de una función de dispersión simple. La tabla de dispersión contiene un puntero a la tabla de páginas invertida, que contiene a su vez las entradas de la tabla de páginas. Con esta estructura, hay una entrada en la tabla de dispersión y la tabla de páginas invertida por cada página de memoria real en lugar de una por cada página virtual. Así pues, se necesita una parte fija de la memoria real para las tablas, sin reparar en el número de procesos o de páginas virtuales soportados. Puesto que más de una dirección de memoria pueden corresponderse con la misma entrada de la tabla de dispersión, para gestionar las colisiones se emplea una técnica de encadenamiento. La técnica de dispersión genera normalmente cadenas cortas, de dos a tres entradas cada una.

Traducir la dirección virtual a una dirección real supone hacer referencia a una entrada de la tabla de páginas, que puede estar en la TLB, en memoria principal o en disco. La palabra referenciada puede estar en cache, en memoria principal o en disco. En este último caso, debe cargarse en memoria principal la página que contiene la palabra y su bloque en la cache. Además, se debe actualizar la entrada de la tabla de páginas para dicha página.

17.- Suponga que la tabla de páginas para un proceso que se está ejecutando es la que se muestra a continuación:



Asumiendo que:

* El tamaño de la pagina es de 512 bytes
* Cada dirección de memoria referencia 1 byte
* Los marcos se encuentras contiguos y en orden en memoria (0, 1, 2.. ) a partir de la dirección real 0.

¿Qué dirección física, si existe, correspondería a cada una de las siguientes direcciones virtuales? (No gestione ningún fallo de página, si se produce)

a) 1052

b) 2221

c) 5499

d) 3101

18.- Tamaño de la Página:

La selección del tamaño de la página influye de manera directa sobre el funcionamiento de la memoria virtual. Compare las siguientes situaciones con respecto al tamaño de página, indicando ventajas y desventajas:

* Un tamaño de página pequeño.
* Un tamaño de página grande.

MV con Paginación

Tamaño de Pagina

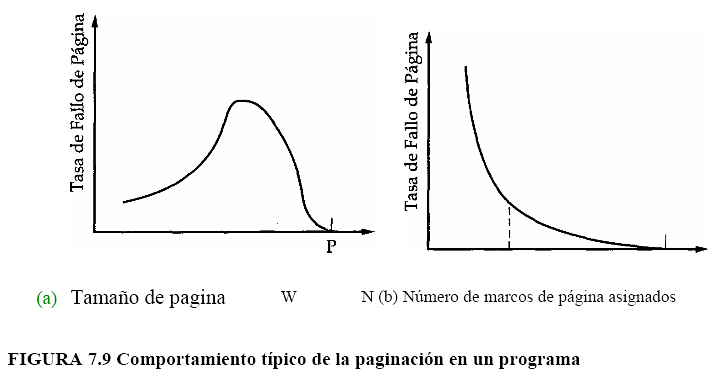
* Pequeño
  + Menor Fragmentación Interna.
  + Más paginas requeridas por proceso 🡺 Tablas de páginas más grandes.
  + Más paginas pueden residir en memoria
* Grande
  + Mayor Fragmentación interna
  + La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente 🡺 Mas rápido mover páginas hacia la memoria principal.

***Tamaño de Página***

Una decisión importante de diseño del hardware es el tamaño de página que se va a usar. Hay varios factores que considerar. Uno es la fragmentación interna. Sin duda, cuanto menor sea el tamaño de página, menor será la cantidad de fragmentación interna. Para optimizar el uso de la memoria principal, es positivo reducir la fragmentación interna. Por otro lado, cuanto menor sea la página, mayor será el número de páginas que se necesitan por proceso. Un número mayor de páginas por proceso significa que las tablas de páginas serán mayores. Para programas grandes, en un entorno multiprogramado intensivo, esto puede significar que una gran parte de las tablas de páginas de los procesos activos deban estar en memoria virtual, no en memoria principal. Así pues, pueden suceder dos fallos de página para una única referencia a memoria: primero, para traer la parte necesaria de la tabla de páginas y, segundo, para traer la página del proceso. Otro factor radica en que las características físicas de la mayoría de los dispositivos de memoria secundaria, que son de rotación, son propicias a tamaños de página mayores, puesto que así se obtiene una transferencia por bloques de datos que es más eficiente.

Para complicar la cuestión se tiene el efecto que tiene el tamaño de página en el porcentaje de fallos de página. Este comportamiento se representa, en líneas generales, en la figura 7.9a y se basa en el principio de cercanía. Si el tamaño de página es muy pequeño, normalmente estarán disponibles en memoria principal un gran número de páginas para cada proceso. Después de un tiempo, todas las páginas en memoria contendrán parte de las referencias más recientes del proceso. Así pues, la tasa de fallos de página será menor. Cuando se incrementa el tamaño de la página, cada página individual contendrá posiciones cada vez más distantes de cualquier referencia reciente. Así pues, se atenúa el efecto del principio de cercanía y comienza a aumentar la tasa de fallos de página. Sin embargo, la tasa de fallos de página comenzará a bajar cuando, finalmente, el tamaño de página se aproxime al tamaño de todo el proceso (punto P del diagrama). Cuando una única página abarca todo el proceso, no hay fallos de página.

Una dificultad más es que la tasa de fallos de página viene determinada también por el número de marcos asignados a un proceso. La figura 7.9b demuestra que, para un tamaño de página fijo, la tasa de fallos baja conforme sube el número de páginas contenidas en memoria principal. Así pues, una política del software (la cantidad de memoria asignada a cada proceso) afectará a una decisión de diseño del hardware.



Por último, el diseño del tamaño de página está relacionado con el tamaño de la memoria física principal. Las implementaciones de memoria virtual en la mayoría de los sistemas actuales fueron diseñadas bajo el supuesto de que la memoria física no es muy grande, normalmente comprendida en un rango de 4M a 256M bytes. Sin embargo, en los próximos años se pueden esperar que aparezcan tamaños de memoria física mayores en estaciones de trabajo y grandes máquinas.

19.- Asignación de marcos a un proceso (Conjunto de trabajo o Working Set):

Con la memoria virtual paginada, no se requiere que todas las páginas de un proceso se encuentren en memoria. El SO debe controlar cuantas páginas de un proceso puede tener en la memoria principal. Existen 2 políticas que se pueden utilizar:

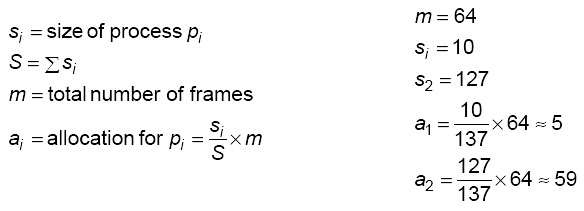
* Asignación Fija
* Asignación Dinámica.

a) Describa como trabajan estas 2 políticas.

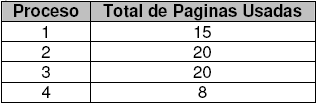
MV con Paginación

Asignación de Marcos

* Cuántas páginas de un proceso se pueden encontrar en memoria?
* Asignación Dinámica
  + El número de marcos para cada proceso varía
* Asignación Fija
  + Número fijo de marcos para cada proceso
* Asignación equitativa – Ejemplo: si tengo 100 frames y 5 procesos, 20 frames para cada proceso
* Asignación Proporcional: Se asigna acorde al tamaño del proceso.



b) Dada la siguiente tabla de procesos y las páginas que ellos ocupan, y teniéndose 40 marcos en la memoria principal, cuantos marcos le corresponderían a cada proceso si se usa la técnica de Asignación Fija:



i) Reparto Equitativo

40 marcos

4 procesos 40 / 4 ₧10 marcos para cada proceso.

ii) Reparto Proporcional

Proceso 1: 15 / 63 \* 40 🡺 9.52

Proceso 2: 20 / 63 \* 40 🡺 12.69

Proceso 3: 20 / 63 \* 40 🡺 12.69

Proceso 4: 8 / 63 \* 40 🡺 5.07

c) ¿Cuál de los 2 repartos usados en b) resulto más eficiente? ¿Por qué?

20.- Reemplazo de páginas (selección de una víctima):

¿Qué sucede cuando todos los marcos en la memoria principal están usados por las páginas de los procesos y se produce en fallo de página? El SO debe seleccionar una de las páginas que se encuentra en memoria como víctima, y ser reemplazada por la nueva página que produjo el fallo.

Considere los siguientes algoritmos de selección de victimas básicos:

* LRU
* FIFO
* OPT (Optimo)
* Segunda Chance

a) Clasifique estos algoritmos de malo a bueno de acuerdo a la tasa de fallos de página que se obtienen al utilizarlos.

***MV con Paginación***

***Reemplazo de páginas***

* ***Optimo***
  + Selecciona la pagina cuyo próxima referencia se encuentra lejana a la actual
  + Imposible de implementar 🡺 No se conoce los futuros eventos
* ***LRU (Least Recently Used)***
  + Reemplaza las páginas que no fue referenciada por más tiempo.
  + Cada página debe tener información del instante de su última referencia 🡺 Se requiere mas overhead para buscarla.
* ***FIFO (First-in, first-out)***
  + Trata a los frames en uso como una cola circular.
  + Simple de implementar.
  + La página más vieja en la memoria es reemplazada.
  + La página puede ser necesitada pronto.
* ***2da. Chance***
  + Se utiliza un Bit adicional 🡺 Bit de referencia
  + Cuando la página se carga en memoria, el bit R se pone en 0
  + Cuando la página es referenciada el bit R se coloca en 1.
  + La víctima se busca en orden FIFO. Se selecciona la primer pagina cuyo bit R esta en 0.
  + Mientras se busca la victima cada bit R que tiene el valor 1, se cambia en 0.

***Algoritmos básicos***

Existen ciertos algoritmos básicos que se emplean para la selección de una página a reemplazar:

• Óptima

• Usada hace más tiempo (LRU, *Least Recently U sed)*

• Primera en entrar, primera en salir (FIFO, *First-In, First-Out)*

***•* De** reloj

La política **óptima** selecciona para reemplazar la página que tiene que esperar una mayor cantidad tiempo hasta que se produzca la referencia siguiente. Se puede demostrar que esta política genera el menor número de fallos de página. Sin duda, este algoritmo resulta imposible de implementar, puesto que requiere que el sistema operativo tenga un conocimiento exacto de los sucesos futuros. Sin embargo, sirve como un estándar con el que comparar los otros algoritmos.

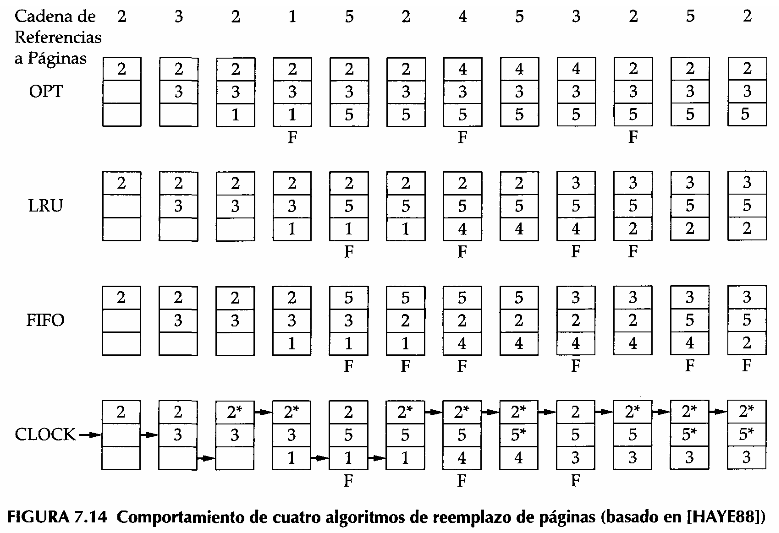
La figura 7.14 ofrece un ejemplo de la política óptima. El ejemplo supone una asignación constante de tres marcos para el proceso. La ejecución del proceso hace referencia a cinco páginas distintas. La cadena de referencias a las páginas durante la ejecución del programa es:

232 152453252

lo que significa que la primera referencia es a la página 2, la segunda a la 3 y así sucesivamente. El algoritmo óptimo origina tres fallos de página después de haber llenado los marcos asignados.

La política de la **usada hace más tiempo** (LRU, *Least Recently U sed)* reemplaza la página de memoria que no ha sido referenciada desde hace más tiempo. Debido al principio de cercanía, ésta debería ser la página con menor probabilidad de ser referenciada en un futuro cercano. De hecho, la política LRU afina casi tanto como la política óptima. El problema de este método es su dificultad de implementación. Una solución sería etiquetar cada página con el instante de su última referencia; esto tendría que hacerse para cada referencia a memoria, tanto para instrucciones como datos. Incluso si el hardware respaldara este esquema, la sobrecarga resultaría tremenda. Como alternativa, se podría mantener una pila de referencias a página, aunque también con un coste elevado.

La figura 7.14 muestra un ejemplo del comportamiento del LRU, utilizando la misma cadena de referencias a páginas del ejemplo de la política óptima. En este caso, se producen cuatro fallos de página.



La política de **primera en entrar, primera en salir (FIFO)** trata los marcos asignados a un proceso como un buffer circular y las páginas se suprimen de memoria según la técnica de espera circular *(round-robín).* Todo lo que se necesita es un puntero que circule a través de los marcos del proceso. Esta es, por tanto, una de las políticas de reemplazo más sencillas de implementar. La lógica que hay detrás de esta elección, además de su sencillez, es reemplazar la página que ha estado más tiempo en memoria: una página introducida en memoria hace mucho tiempo puede haber caído en desuso. Este razonamiento será a menudo incorrecto, porque habrá regiones de programa o de datos que son muy usadas a lo largo de la vida de un programa. Con el algoritmo FIFO, estas páginas se cargarán y expulsarán repetidas veces.

Continuando con el ejemplo de la figura 7.14, la política FIFO genera seis fallos de página. Nótese que la LRU se da cuenta de que las páginas 2 y 5 se referencian más frecuentemente que las otras, mientras que la FIFO no lo hace.

Si bien la política LRU se acerca mucho a la política óptima, es difícil de implementar e impone una sobrecarga significativa. Por otro lado, la política FIFO es muy simple de implementar, pero su rendimiento es relativamente pobre. A lo largo de los años, los diseñadores de sistemas operativos han probado una serie de algoritmos para aproximarse al rendimiento de la LRU sin introducir mucha sobrecarga. Muchos de esos algoritmos son variantes de un esquema denominado **política del reloj.**

La forma más simple de la política del reloj requiere asociar un bit adicional a cada marco, denominado *bit de uso.* Cuando se carga una página por primera vez en un marco de memoria, el bit de uso de dicho marco se pone a cero. Cuando se hace referencia a la página posteriormente (después de la referencia que generó el fallo de página), el bit de uso se pone a 1. Para el algoritmo de reemplazo de páginas, el conjunto de marcos candidatos a ser reemplazados se considera como un buffer circular con un puntero asociado. El alcance es local si los candidatos son de un solo proceso; el alcance es global si los candidatos provienen de toda la memoria principal. Al reemplazar una página, se hace que el puntero señale al siguiente marco del buffer. Cuando llega el momento de reemplazar una página, el sistema operativo recorre el buffer buscando un marco con el bit de uso a 0. Cada vez que se encuentra un marco con el bit de uso a 1, lo pone a 0. Si algún marco del buffer tiene el bit de uso a 0 al comienzo de la búsqueda, se elige para reemplazar el primero que se haya encontrado. Si todos los marcos tienen el bit de uso puesto a 1, el puntero dará una vuelta completa al buffer, poniendo todos los bits a 0 y se detendrá en la posición inicial, reemplazando la página de dicho marco. Es posible comprobar que esta política es similar a la FIFO, excepto que cualquier marco con el bit de uso a 1 se descarta en el algoritmo. La política se denomina política del reloj porque se pueden imaginar los marcos dispuestos en círculo. Varios sistemas operativos han empleado una variación de esta simple política, como por ejemplo Multics.

El comportamiento de la política del reloj se muestra en la figura 7.14. La presencia de un asterisco indica que el bit de uso correspondiente es igual a 1 y la flecha indica la posición actual del puntero. Así pues, en la tercera unidad de tiempo, se hace referencia a la página 2 y se activa su bit de uso. En la cuarta unidad de tiempo, se carga una página en el tercer marco y el puntero se sitúa en el primero. Sin embargo, puesto que el bit de uso de este marco está activo, no se usará como marco de reemplazo en el quinto instante de tiempo. Nótese que la política del reloj es adecuada para proteger a los marcos 2 y 5 del reemplazo. De hecho, en este ejemplo, la política del reloj supera el rendimiento del LRU. En general, no es éste el caso.

La figura 7.16 muestra el resultado de un experimento que compara los cuatro algoritmos que se han estudiado; se supone que el número de marcos asignado a un proceso es fijo. Los resultados están basados en la ejecución de 0,25 x 10^6 referencias en un programa en FORTRAN usando un tamaño de página de 256 palabras. Baer realizó la prueba con asignaciones de 6, 8, 10, 12 y 14 marcos. Las diferencias entre las cuatro políticas son más notorias con asignaciones pequeñas. Con FIFO se rondaba un factor de 2 veces peor que el óptimo

El algoritmo del reloj también se ha comparado con los demás algoritmos cuando se emplea asignación variable y un alcance del reemplazo tanto global como local.

Nuevamente, se determinó que el rendimiento del algoritmo del reloj se aproxima al del LRU.

El algoritmo del reloj puede hacerse más potente incrementando el número de bits empleados. En todos los procesadores que ofrecen paginación, se asocia un bit de modificación con cada página en memoria principal y, por tanto, con cada marco. Este bit es necesario para que, cuando se modifica una página, no se reemplace hasta volverla a escribir en memoria secundaria. Es posible aprovechar este bit en el algoritmo del reloj del siguiente modo. Si se tienen en cuenta el bit de uso y el de modificación, cada marco estará en una de las siguientes cuatro categorías:

• No accedido recientemente y sin modificar *(u = 0; m =* 0)

• Accedido recientemente y sin modificar (*u* = 1; *m* = 0)

• No accedido recientemente y modificado (w = 0; *m =* 1)

• Accedido recientemente y modificado *(u =* 1; *m =* 1)

Con esta clasificación, el algoritmo del reloj se comporta de la forma siguiente:

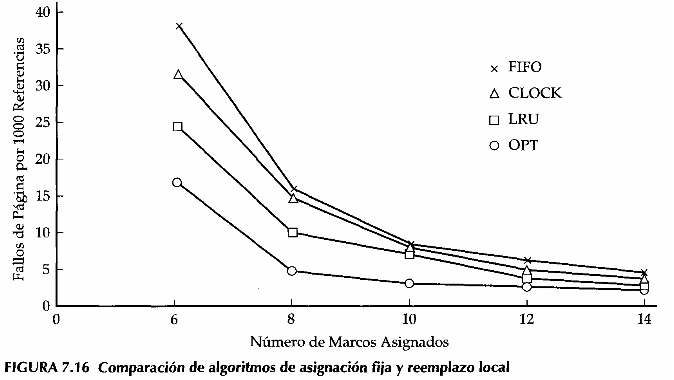
1. Comenzando en la posición actual del puntero, recorrer el buffer de marcos. Durante este recorrido, no cambiar el bit de uso. Se selecciona para reemplazo el primer marco encontrado con (u = 0; *m =* 0).

2. Si falla el paso 1, recorrer de nuevo buscando marcos con *(u =* 0; *m =* 1). Se selecciona para reemplazar el primer marco encontrado. Durante este recorrido, se pone a O el bit de uso de cada marco por el que se pasa.

3. Si falla el paso 2, el puntero habrá retomado a su posición original y todos los marcos del conjunto tendrán el bit de uso a 0. Repetir el paso 1. Esta vez, se encontrará un marco para reemplazar.

En resumen, el algoritmo de reemplazo de páginas recorre todas las páginas del buffer buscando una que no se haya modificado desde que fue cargada y a la que no se haya accedido recientemente. Tal página es una buena candidata para su reemplazo y tiene la ventaja de que, como no está modificada, no hace falta volver a escribirla en memoria secundaria. Si no se encuentra ninguna candidata en la primera vuelta, el algoritmo recorre el buffer de nuevo buscando una página modificada a la que no se haya accedido recientemente. Incluso aunque dicha página deba escribirse para reemplazarla, debido al principio de cercanía, no será necesaria de nuevo durante algún tiempo. Si esta segunda pasada falla, todos los marcos del buffer no habrán tenido accesos recientes y se llevará a cabo una tercera pasada.

Esta estrategia es la usada en el esquema de memoria virtual del Macintosh. La ventaja de este algoritmo sobre el algoritmo simple del reloj es que las páginas que no han sido cambiadas tienen preferencia para reemplazarse. Puesto que una página modificada debe escribirse al disco antes de ser reemplazada, se produce un ahorro de tiempo evidente.



b) Analice su funcionamiento. ¿Como los implementaría?

c) Sabemos que la pagina a ser reemplaza puede estar modificada. ¿Qué acciones debe llevar el SO cuando se encuentra ante esta situación?

En todos los procesadores que ofrecen paginación, se asocia un bit de modificación con cada página en memoria principal y, por tanto, con cada marco. Este bit es necesario para que, cuando se modifica una página, no se reemplace hasta volverla a escribir en memoria secundaria.

Otro bit de control necesario en la entrada de la tabla de páginas es el bit de modificación (M), para indicar si el contenido de la página correspondiente se ha alterado desde que la página se cargó en memoria principal. Si no ha habido cambios, no es necesario escribir la página cuando sea sustituida en el marco que ocupa actualmente.

21.- Alcance del reemplazo

Al momento de tener que seleccionar una página víctima, el SO puede optar por 2 políticas a utilizar:

* Reemplazo local
* Reemplazo global

1. Describa como trabajan estas 2 políticas.

MV con Paginación

Alcance del Reemplazo

* Reemplazo Global
  + El fallo de página de un proceso puede reemplazar la página de cualquier proceso.
  + El proceso no controla su tasa de page-faults
  + Puede tomar sólo frames de otro proceso aumentando la cantidad de frames asignados a él.
  + Un proceso de alta prioridad podría tomar los frames de un proceso de menor prioridad.
* Reemplazo Local
  + El fallo de página de un proceso solo puede reemplazar sus propias páginas.
  + No cambia la cantidad de frames asignados
  + El proceso puede determinar cuál es su tasa de page-faults.
  + Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos.

El alcance de un reemplazo puede clasificarse en global o local. Ambos tipos de políticas son activadas por un fallo de página que se produce cuando no hay marcos libres. Para seleccionar la página a reemplazar, una **política de reemplazo local** escoge únicamente de entre las páginas residentes del proceso que originó el fallo de página. Una **política de reemplazo global** considera todas las páginas en memoria como candidatas para reemplazar, independientemente del proceso al que pertenezcan. Aunque las políticas locales son más fáciles de analizar, no hay ninguna evidencia de que se comporten mejor que las políticas globales, las cuales son atrayentes por su simplicidad de implementación y su mínimo coste.

Existe una relación entre el alcance del reemplazo y el tamaño del conjunto residente. Un conjunto residente fijo implica una política de reemplazo local: para mantener constante el tamaño del conjunto residente, una página suprimida de memoria principal debe reemplazarse por otra del mismo proceso. Una política de asignación variable puede, sin duda, emplear una política de reemplazo global: el reemplazo de una página de un proceso en memoria principal por otra provoca que la asignación de un proceso crezca en una página y la de otro disminuya también en una. También se verá que la asignación variable y el reemplazo local es una combinación válida.

b) ¿Es posible utilizar la política de “Asignación Fija” de marcos junto con la política de “Reemplazo Global? Justifique.

**Asignación fija y alcance local**

En este caso, se tiene un proceso que ejecuta en memoria principal con un número fijo de páginas. Cuando se produce un fallo de página, el sistema operativo debe elegir la página a reemplazar entre las de dicho proceso que están actualmente en memoria.

Con una política de asignación fija, es necesario decidir por anticipado la cantidad de memoria asignada a un proceso. Esta decisión puede hacerse en función del tipo de aplicación y de la cantidad solicitada por el programa. Las desventajas de esta solución son dos: si la asignación tiende a ser demasiado pequeña, se producirá un alto porcentaje de fallos de página, haciendo que el sistema multiprogramado al completo funcione lentamente. Si la asignación tiende a ser innecesariamente grande, habrá muy pocos programas en memoria principal y el procesador estará desocupado un tiempo considerable o bien se consumirá un tiempo importante en intercambio.

**Asignación variable y alcance global**

Esta combinación es quizá la más sencilla de implementar y ha sido adoptada por un buen número de sistemas operativos. En un instante dado, en memoria principal habrá varios procesos, cada uno de ellos con un cierto número de marcos asignados. Normalmente, el sistema operativo también 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 carga la página. Así pues, los procesos que producen fallos de página incrementan gradualmente su tamaño, lo que ayuda a reducir el número global de fallos de página en el sistema.

La dificultad de este método está en la elección del reemplazo. Cuando no hay marcos libres, el sistema operativo debe elegir una página que esté en memoria para reemplazar. La selección se realiza entre todas las páginas de memoria, excepto los marcos bloqueados, como los del núcleo. La página elegida puede pertenecer a cualquier proceso residente; no hay ninguna disciplina que determine el proceso que debe perder una página de su conjunto residente. Es más, la selección del proceso que sufre la reducción en el conjunto residente puede que no sea óptima.

Una forma de contrarrestar los problemas potenciales de rendimiento de una política de asignación variable y alcance global es el almacenamiento intermedio de páginas. De esta manera, la elección de la página a reemplazar se hace menos significativa, ya que la página puede ser recuperada si se produce una referencia antes de que se escriba el bloque con el siguiente grupo de páginas.

**Asignación variable y alcance local**

Otro método de asignación variable intenta superar los problemas de la estrategia de alcance global. La estrategia de asignación variable y alcance local puede resumirse como sigue:

1. Cuando se carga un nuevo proceso en memoria, se le asigna cierto número de marcos en función del tipo de aplicación, las necesidades del programa u otros criterios. La asignación puede cubrirse tanto con paginación previa como con paginación por demanda.

2. Cuando se produce un fallo de página, se selecciona la página a reemplazar de entre las del conjunto residente del proceso que sufre el fallo.

3. De vez en cuando, se vuelve a evaluar la asignación otorgada al proceso y se aumenta o disminuye para mejorar el rendimiento global.

Con esta estrategia, se meditará la decisión de aumentar o disminuir el tamaño del conjunto residente y ésta se hará en función de una valoración de las posibles demandas futuras de los procesos activos. Debido a esta evaluación, la estrategia es más compleja que la simple política de reemplazo global. Sin embargo, puede obtenerse un rendimiento mejor.

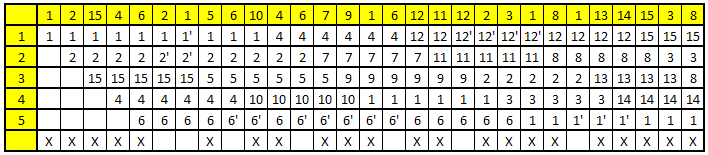
Los elementos clave de la asignación variable con alcance local son los criterios empleados para determinar el tamaño del conjunto residente y el momento de los cambios. Una estrategia específica que ha recibido mucha atención en la bibliografía se conoce como **estrategia del conjunto de trabajo.** Si bien una estrategia pura del conjunto de trabajo sería difícil de implementar, puede ser útil como base para las comparaciones.

22.- Considere la siguiente secuencia de referencias de páginas:

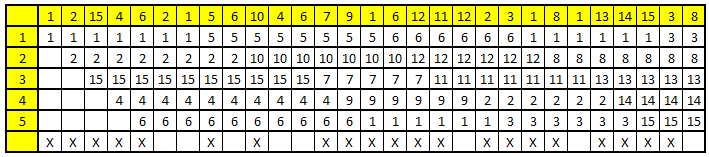
1, 2, 15, 4, 6, 2, 1, 5, 6, 10, 4, 6, 7, 9, 1, 6, 12, 11, 12, 2, 3, 1, 8, 1, 13, 14, 15, 3, 8

a) Si se disponen de 5 marcos. ¿Cuántos fallos de página se producirán si se utilizan las siguientes técnicas de selección de victima? (Considere una política de Asignación Dinámica y Reemplazo Global)

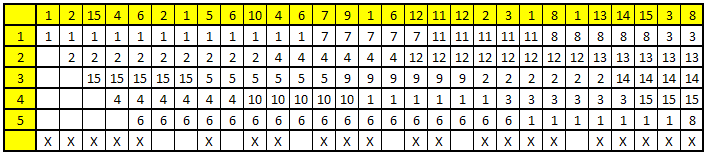
i) Segunda Chance



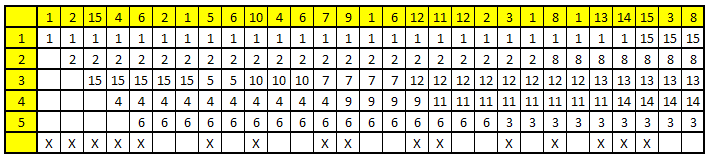
ii) FIFO



iii) LRU



iv) OPT



b) Suponiendo que cada atención de un fallo se pagina requiere de 0,1 seg. Calcular el tiempo consumido por atención a los fallos de páginas para los algoritmos de a).

23.- Sean los procesos A, B y C tales que necesitan para su ejecución las siguientes páginas:

* A: 1, 3, 1, 2, 4, 1, 5, 1, 4, 7, 9, 4
* B: 2, 4, 6, 2, 4, 1, 8, 3, 1, 8
* C: 1, 2, 4, 8, 6, 1, 4, 1

Si la secuencia de ejecución es tal que los procesos se ejecutan en la siguiente secuencia:

1. B demanda 2 páginas

2. A demanda 3 páginas

3. C demanda 2 páginas

4. B demanda 3 páginas

5. A demanda 3 páginas

6. C demanda 2 páginas

7. B demanda 2 páginas

8. C demanda 4 páginas

9. A demanda 3 páginas

10. B demanda 3 páginas

11. C termina

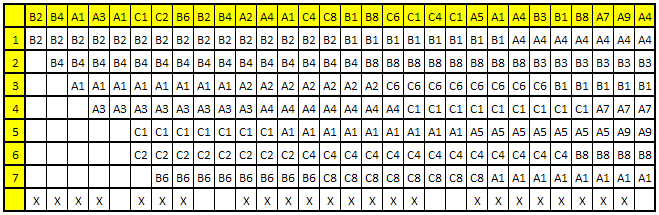
12. A demanda 3 páginas

13. B termina

14. A termina

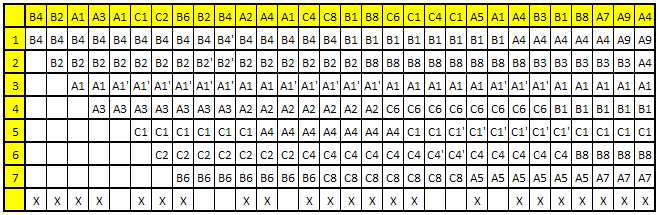
a) Considerando una política de Asignación Dinámica y Reemplazo Global y disponiéndose de 7 marcos. ¿Cuántos fallos de página se producirán si se utiliza la técnica de selección de victimas?:

i) LRU



🡺 24 fallos.

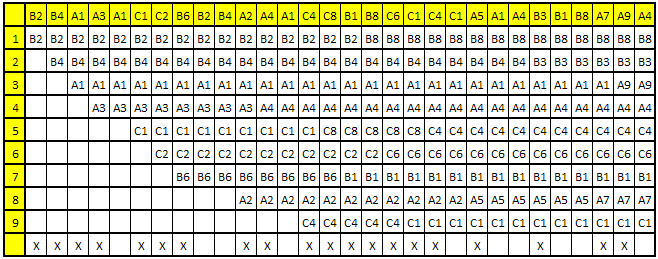
ii) Segunda Chance



🡺 23 fallos.

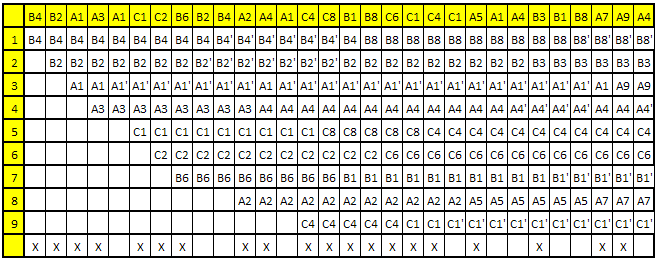
b) Considerando una política de Asignación Fija con reparto equitativo y Reemplazo Local y disponiéndose de 9 marcos. ¿Cuántos fallos de página se producirán si se utiliza la técnica de selección de victimas?:

i) LRU



🡺 20 fallos.

ii) Segunda Chance



🡺 20 fallos.

24.- Sean los procesos A, B y C tales que necesitan para su ejecución las siguientes páginas:

* A: 1, 2, 1, 7, 2, 7, 3, 2
* B: 1, 2, 5, 2, 1, 4, 5
* C: 1, 3, 5, 1, 4, 2, 3

Si la secuencia de ejecución es tal que los procesos se ejecutan en la siguiente manera:

1. C demanda 1 página

2. A demanda 2 páginas

3. C demanda 1 página

4. B demanda 1 página

5. A demanda 1 página

6. C modifica la página 1

7. B demanda 2 páginas

8. A demanda 1 página

9. C demanda 1 página

10. B modifica la página 2

11. A modifica la página 2

12. B demanda 2 páginas

13. A demanda 1 página

14. B demanda 2 páginas

15. C demanda 2 páginas

16. C demanda 1 página

17. A demanda 1 página

18. B termina

19. A demanda 2 páginas

20. C demanda 1 página

21. A termina

22. C termina

Considerando una política de Asignación Dinámica y Reemplazo Global y disponiéndose de 7 marcos, debiéndose guardar 1 marco para la gestión de descarga asincrónica de páginas modificadas ¿Cuántos fallos de página se producirán si se utiliza la técnica de selección de víctima?:

a) Segunda Chance

b) FIFO

c) LRU

25.- Hiperpaginación (Trashing)

a) ¿Qué es?

b) ¿Cuáles pueden ser los motivos que la causan?

c) ¿Cómo la detecta el SO?

d) Una vez que lo detecta, ¿qué acciones puede tomar el SO para eliminar este problema?

Considérese un proceso grande formado por un programa largo y un conjunto de series de datos. Durante un corto periodo, la ejecución puede estar reducida a una pequeña sección del programa (por ejemplo, una subrutina) y acceder sólo a una o dos series de datos. Si esto es así, sería un claro desperdicio cargar docenas de fragmentos para el proceso cuando se van a usar sólo unos pocos antes de que pase a estar suspendido o se descargue. Se puede aprovechar mejor la memoria cargando tan sólo unos pocos fragmentos. Además, si el programa se bifurca a una instrucción o hace referencia a datos de un fragmento que no está en memoria, se producirá un fallo de página. Este fallo le dice al sistema operativo que traiga el fragmento deseado.

De este modo, en un instante dado, en memoria sólo se tienen unos pocos fragmentos de un proceso dado y, por tanto, se pueden mantener más procesos en memoria. Es más, se ahorra tiempo, porque los fragmentos que no se usan no se cargan ni se descargan de memoria. Sin embargo, el sistema operativo debe saber cómo gestionar este esquema. En un estado estable, prácticamente toda la memoria principal estará ocupada con fragmentos de procesos, por lo que el procesador y el sistema operativo tendrán acceso directo a la mayor cantidad de procesos posible. Así pues, cuando el sistema operativo traiga a memoria un fragmento, deberá expulsar otro. Si expulsa un fragmento justo antes de ser usado, tendrá que traer de nuevo el fragmento de manera casi inmediata. Demasiados intercambios de fragmentos conducen a lo que se conoce como **hiperpaginación** *(thrashing):* El procesador consume más tiempo intercambiando fragmentos que ejecutando instrucciones de usuario. Las formas de evitar la hiperpaginación fueron una de las áreas de investigación más importantes de los 70 y llevaron a un buen número de algoritmos complejos pero efectivos. En esencia, el sistema operativo intenta adivinar, en función de la historia reciente, qué fragmentos se usarán con menor probabilidad en un futuro próximo.

* Thrashing (hiperpaginación).
  + A groso modo, decimos que un sistema está en thrashing cuando pasa más tiempo paginando que ejecutando procesos.

Si un proceso cuenta con todos los frames que necesita, no habría thrashing!

* Existen técnicas para evitarlo:
  + Estrategia de Working Set basado en el modelo de localidad.

**Ciclo del thrashing**

1) El SO monitorea el uso de la CPU.

2) Si hay baja utilización 🡺 aumenta el grado de multiprogramación.

3) Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos.

4) Un proceso necesita más frames. Comienzan los page-faults y robo de frames a otros procesos.

5) Por swapping de páginas, y encolamiento en dispositivos, baja el uso de la CPU.

6) Vuelve a 1).

**Participación del scheduler de CPU, en thrashing**

1) Cuando se decrementa el uso de la CPU, el scheduler long term aumenta el grado de multiprogramación.

2) El nuevo proceso inicia nuevos pagefaults, y por lo tanto, más actividad de paginado.

3) Se decrementa el uso de la CPU

4) Vuelve a 1).

**Control del thrashing**

* Se puede limitar el thrashing usando algoritmos de reemplazo local.
* Con este algoritmo, si un proceso entra en thrashing no roba frames a otros procesos.
* Si bien perjudica la performance del sistema, es controlable.

**Conclusión sobre thrashing**

* Si un proceso cuenta con todos los frames que necesita, no habría thrashing.
* Veremos algunas técnicas como la estrategia de Working Set, con el modelo de localidad.

**Control de carga**

El control de carga consiste en determinar el número de procesos que pueden estar en memoria principal, lo que ha venido en llamarse grado de multiprogramación. La política de control de carga es crítica para la efectividad de la gestión de memoria. Si, en un instante dado, hay pocos procesos residentes en memoria, habrá muchas ocasiones en las que todos los procesos estén bloqueados y se consumirá mucho tiempo en el intercambio. Por otro lado, si hay demasiados procesos residentes, el tamaño medio del conjunto residente de cada proceso no será el adecuado y se producirán frecuentes fallos de página. El resultado es una situación conocida como *hiperpaginación (thrashing).*

***Grado de multiprogramación***

Cuando el grado de multiprogramación supera un pequeño valor, se podría esperar que la utilización del procesador aumentara, puesto que hay menos posibilidades de que todos los procesos estén bloqueados. Sin embargo, se alcanza un punto en el que el conjunto residente en promedio no es adecuado. En este punto, el número de fallos de página se eleva drásticamente y la utilización del procesador se desploma.

Hay varias formas de abordar este problema. Los algoritmos del conjunto de trabajo o de frecuencia de fallos de página incorporan implícitamente el control de carga. Sólo pueden ejecutar aquellos procesos cuyos conjuntos residentes sean suficientemente grandes. Dado un tamaño exigido para el conjunto residente de cada proceso activo, la política determina automática y dinámicamente el número de programas activos.

Otro método, propuesto por Denning y sus colaboradores, se conoce como el "criterio *L =* 5"', que ajusta el grado de multiprogramación de forma que el tiempo medio entre fallos sea igual al tiempo medio exigido para procesar un fallo de página. Algunos estudios de rendimiento indican que éste es el punto en el que el uso del procesador alcanza un máximo. Una política con un efecto similar, es el "criterio del 50%", que intenta mantener el uso del dispositivo de paginación aproximadamente a un 50%. Otros estudios de rendimiento también indican que éste es un punto de máximo aprovechamiento del procesador.

Otro método consiste en adaptar el algoritmo del reloj para reemplazo de páginas para que permita incorporar una supervisión de la velocidad en la que el puntero recorre el buffer circular de marcos. Si la velocidad está por debajo de un umbral inferior determinado, es que se produce una de estas situaciones o ambas a la vez:

1. Se producen pocos fallos de página, lo que genera pocas peticiones de avance del puntero.

2. Para cada petición, el puntero explora un número medio de marcos pequeño, lo que indica que hay muchas páginas de memoria sin referenciar y que están listas para reemplazar.

En ambos casos, el grado de multiprogramación puede aumentarse con seguridad. Por otro lado, si la velocidad de exploración del puntero excede un umbral superior, quiere decirse que la tasa de fallos de página es alta o que es difícil encontrar páginas para reemplazar, lo que implica que el grado de multiprogramación es demasiado alto.

26.- Considere un sistema cuya memoria principal se administra mediante la técnica de paginación por demanda que utiliza un dispositivo de paginación, algoritmo de reemplazo global LRU y una política de asignación que reparte marcos equitativamente entre los procesos. El nivel de multiprogramación es actualmente, de 4.

Ante las siguientes mediciones:

Analizar:

1. ¿Qué sucede en cada caso?
2. ¿Puede incrementarse el nivel de multiprogramación para aumentar el uso de la CPU?
3. ¿La paginación está siendo útil para mejorar el rendimiento del sistema?

a) Uso de CPU del 13%, uso del dispositivo de paginación del 97%.

1. Hay mas procesos paginando que ejecutándose.
2. No puede incrementarse el nivel de multiprogramación porque produciría un aumento del uso del dispositivo de paginación y bajaría todavía más el aprovechamiento de la CPU.
3. No está siendo útil 🡺 se está produciendo HIPERPAGINACIÒN.

b) Uso de CPU del 87%, uso del dispositivo de paginación del 3%.

1. Se está aprovechando de forma óptima el procesador, sin producir demasiados PAGE-FAULTS.
2. Si se incrementa el nivel de multiprogramación, podría pasar que los procesos, al ser mayor su cantidad, empiecen a pelear por los marcos y aumente la cantidad de PAGE-FAULTS reduciendo el aprovechamiento de la CPU.
3. Si, está siendo útil ya que no hay PAGE-FAULTS en exceso.

c) Uso de CPU del 13%, uso del dispositivo de paginación del 3%.

1. Bajo aprovechamiento de la CPU con pocos fallos de página.
2. Se podría aumentar, de forma controlada, el nivel de multiprogramación para aumentar el uso de la CPU.
3. Hay poco porcentaje de paginación (PAGE-FAULTS), por lo tanto ayuda a aumentar el rendimiento del sistema.

27.- Considere un sistema cuya memoria principal se administra mediante la técnica de paginación por demanda. Considere las siguientes medidas de utilización:

* Utilización del procesador: 20%
* Utilización del dispositivo de paginación: 97,7%
* Utilización de otros dispositivos de E/S: 5%

Cuáles de las siguientes acciones pueden mejorar la utilización del procesador:

a) Instalar un procesador más rápido

***b) Instalar un dispositivo de paginación mayor***

c) Incrementar el grado de multiprogramación

d) Instalar más memoria principal

e) Decrementar el quantum para cada proceso

Con **paginación por demanda,** se trae una página a memoria principal sólo cuando se hace referencia a una posición en dicha página. Si los otros elementos de la política de gestión de memoria funcionan adecuadamente, debe ocurrir lo siguiente. Cuando un proceso se ejecute por primera vez, se producirá un aluvión de fallos de página. A medida que se traigan a memoria más páginas, el principio de cercanía hará que la mayoría de las futuras referencias estén en páginas que se han cargado hace poco. Así pues, después de un tiempo, la situación se estabilizará y el número de fallos de página disminuirá hasta un nivel muy bajo.

28.- La siguiente formula describe el tiempo de acceso efectivo a la memoria al utilizar paginación para la implementación de la memoria virtual:

TAE = At + (1 - p) \* Am + p \*(Tf + Am)

Donde:

TAE = tiempo de acceso efectivo

p = taza de fallo de pagina (0 <= p <=1)

Am = tiempo de acceso a la memoria real

Tf = tiempo se atención de una fallo de pagina

At = tiempo de acceso a la tabla de páginas. Es igual al tiempo de acceso a la memoria (Am) si la entrada de la tabla de páginas no se encuentra en la TLB.

Suponga que tenemos una memoria virtual paginada, con tabla de páginas de 1 nivel, y donde la tabla de páginas se encuentra completamente en la memoria. Servir una falla de página tarda 300 nanosegundos si hay disponible un marco vacío o si la página reemplazada no se ha modificado, y 500 nanosegundos si se ha modificado. El tiempo de acceso a memoria es de 20 nanosegundos y el de acceso a la TLB es de 1 nanosegundo

1 nanosegundo == 10^(-9) segundos.

a) Si suponemos una taza de fallos de página de 0,3 y que siempre contamos con un marco libre para atender el fallo ¿Cuál será el TAE si el 50% de las veces la entrada de la tabla de páginas se encuentra en la TLB (hit)?

TAE = At + (1 - p) \* Am + p \*(Tf + Am)

TAE = (1 + 20) / 2 nano + (1 - 0,3) \* 20 nano + 0,3 \* (300 nano + 20 nano)

TAE = 10,5 nano + 0,7 \* 20 nano + 0,3 \* 300 nano

TAE = 0,0000000105 s + 0,000000014 s + 0,000000096 s

TAE = 0,00000011105 segundos

b) Si suponemos una taza de fallos de página de 0,3; que el 70% de las ocasiones la pagina a reemplazar se encuentra modificada. ¿Cuál será el TAE si el 60% de las veces la entrada de la tabla de páginas se encuentra en la TLB (hit)?

TAE = At + (1 - p) \* Am + p \*(Tf + Am)

TAE = (0,6 + 8) nano + (1 - 0,3) \* 20 nano + 0,3 \* ((350 + 90) nano + 20 nano)

TAE = 8,6 nano + 0,7 \* 20 nano + 0,3 \* 460 nano

TAE = 0,0000000086 s + 0,000000014 s + 0,000000138 s

TAE = 0,0000001606 segundos

c) Si suponemos que el 60% de las veces la pagina a reemplazar esta modificada, el 100% de las veces la entrada de la tabla de páginas requerida se encuentra en la TLB (hit) y se espera un TAE menor a 200 nanosegundos. ¿Cuál es la máxima tasa aceptable de fallas de página?

20 nanosegundos = 0,0000002 > 0,00000019

TAE = At + (1 - p) \* Am + p \*(Tf + Am)

0,00000019 = 1 + (1 - p) \* 20 + p \* ((300 + 120) + 20)

0,00000019 = 1 + (1 - p) \* 20 + p \* 440

0,00000019 = 1 + 20 - 20p + 440p

0,00000019 = 21 + 420p

(0,00000019 – 21) / 420 = p

29.- Anomalía de Belady

a) ¿Qué es?

La **anomalía de Belady** es un efecto descubierto y demostrado en [1969](http://es.wikipedia.org/wiki/1969) por el científico de la computación [húngaro](http://es.wikipedia.org/wiki/Hungr%C3%ADa) [Laszlo Belady](http://es.wikipedia.org/w/index.php?title=Laszlo_Belady&action=edit&redlink=1), por el cual es posible tener más [fallos de página](http://es.wikipedia.org/w/index.php?title=Fallo_de_p%C3%A1gina&action=edit&redlink=1) al aumentar el número de marcos en la memoria física utilizando el método [FIFO](http://es.wikipedia.org/wiki/FIFO) como [algoritmo de reemplazo de páginas](http://es.wikipedia.org/wiki/Algoritmos_de_reemplazo_de_p%C3%A1ginas) en sistemas de gestión de [memoria virtual](http://es.wikipedia.org/wiki/Memoria_virtual) con [paginación](http://es.wikipedia.org/wiki/Paginaci%C3%B3n). Antes de esta fecha, se creía que incrementar el número de marcos físicos siempre llevaría a un descenso del número fallo de páginas o, en el peor de los casos, a mantenerlo. Así, pues, antes del descubrimiento de la anomalía de Belady, el algoritmo FIFO era aceptable.

El siguiente es un ejemplo de la anomalía de Belady. Utilizando tres marcos ocurren 9 fallos de página. Aumentando a cuatro marcos obtenemos 10 fallos de página. Los fallos de página están en rojo. El contenido de la memoria principal en cada momento está debajo de cada nueva petición de página.

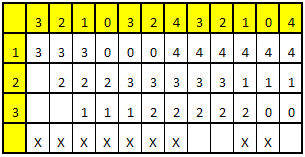
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Peticiones de página** | **3** | **2** | **1** | **0** | **3** | **2** | **4** | **3** | **2** | **1** | **0** | **4** |
| **Página más nueva** | 3 | 2 | 1 | 0 | 3 | 2 | 4 | 4 | 4 | 1 | 0 | 0 |
|  |  | 3 | 2 | 1 | 0 | 3 | 2 | 2 | 2 | 4 | 1 | 1 |
| **Página más antigua** |  |  | 3 | 2 | 1 | 0 | 3 | 3 | 3 | 2 | 4 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Peticiones de página** | **3** | **2** | **1** | **0** | **3** | **2** | **4** | **3** | **2** | **1** | **0** | **4** |
| **Página más nueva** | 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 |
| **Página más antigua** |  |  |  | 3 | 3 | 3 | 2 | 1 | 0 | 4 | 3 | 2 |
| **(rojo indica fallo de página)** | | | | | | | | | | | | |

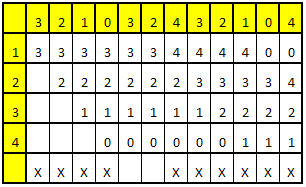
b) Dada la siguiente secuencia de referencias a páginas:

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

I. Calcule la cantidad de fallos de páginas si se cuentan con 3 marcos y se utiliza el algoritmo de reemplazo FIFO



II. Calcule la cantidad de fallos de páginas si se cuentan con 4 marcos y se utiliza el algoritmo de reemplazo FIFO



Analice la situación

Se produce un resultado reflejando la anomalía de Belady ya que el número de fallos de página con cuatro marcos es mayor al número de fallos de páginas con tres marcos.

30.- Considere el siguiente programa:

#define Size 64

int A[Size; Size], B[Size; Size], C[Size; Size];

int register i, j;

for (j = 0; j < Size; j ++)

for (i = 0; i < Size; i++)

C[i; j] = A[i; j] + B[i; j];

Si asumimos que el programa se ejecuta en un sistema que utiliza paginación por demanda para administrar la memoria, donde cada página es de 1Kb. Cada número entero (int) ocupa 4 bytes. Es claro que cada matriz requiere de 16 páginas para almacenarse. Por ejemplo: A[0,0]..A[0,63], A[1,0]..A[1,63], A[2,0]..A[2,63] y

A[3,0]..A[3,63] se almacenara en la primer página.

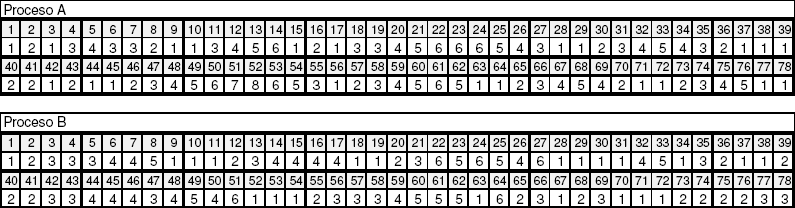
Asumamos que el sistema utiliza un working set de 4 marcos para este proceso. Uno de los 4 marcos es utilizado por el programa y los otros 3 se utilizan para datos (las matrices). También asumamos que para los índices “i” y “j” se utilizan 2 registros, por lo que no es necesario el acceso a la memoria para estas 2 variables.

a) Analizar cuantos fallos de páginas ocurren al ejecutar el programa (considere las veces que se ejecuta

C[i,j] = A[i,j] + B[i,j])

b) Puede ser modificado el programa para minimizar el número de fallos de páginas. En caso de ser posible indicar la cantidad de fallos de fallos de páginas que ocurren.

31.- Considere las siguientes secuencias de referencias a páginas de los procesos A y B, donde se muestra en instante de tiempo en el que ocurrió cada una (1 a 78):



a) Considerando una ventana Δ=5, indique cual sería el conjunto de trabajo de los procesos A y B en el instante 24 (WS A (24) y WS B (24))

b) Considerando una ventana Δ=5, indique cual sería el conjunto de trabajo de los procesos A y B en el instante 60 (WSA (60) y WSB (60))

c) Para los WS obtenidos en el inciso a), si contamos con 8 frames en el sistema ¿Se puede indicar que estamos ante una situación de trashing? ¿Y si contáramos con 6 frames?

d) Considerando únicamente el proceso A, y suponiendo que al mismo se le asignaron inicialmente 4 marcos, donde el de reemplazo de páginas es realizado considerando el algoritmo FIFO. ¿Cuál será la taza de fallos en el instante 38 de páginas suponiendo que la misma se calcula contando los fallos de páginas que ocurrieron en las últimas 10 unidades de tiempo?

e) Para el valor obtenido en el inciso d), si suponemos que el S.O. utiliza como limites superior e inferior de taza de fallos de paginas los valores 2 y 5 respectivamente ¿Qué acción podría tomar el S.O. respecto a la cantidad de

marcos asignados al proceso?