Sistemas Operativos

Unidad 8: Memoria virtual.

"Nunca memorices algo que puedas buscar."

~ Albert Einstein

Memoria Virtual

Introducción

Las técnicas de paginación y segmentación analizadas en la unidad anterior, presentan dos características importantes que facilitaron la implementación de la multiprogramación:

- Todas las referencias a memoria dentro de un proceso se realizan a direcciones lógicas, que deben ser traducidas a direcciones físicas. Esto posibilita que un proceso pueda ser expulsado de la memoria y reubicado nuevamente en otra posición de memoria.
- Un proceso puede dividirse en porciones y las mismas pueden ser cargadas en posiciones de memoria no necesariamente contiguas.

Ahora bien, por los temas vistos hasta ahora sabemos que para poder ser ejecutado, un proceso debe estar cargado en memoria, es decir, totalmente cargado en memoria.

Aclaración	**¿Cómo sabe el sistema operativo la cantidad de memoria inicial que necesita un proceso para cargarlo en memoria y ejecutarlo?** Esta información se encuentra generalmente en el *header* (encabezado) del archivo ejecutable y la misma es generada en el proceso de compilación.

¿Qué pasa si el espacio ocupado por el proceso es mayor que la cantidad de memoria? Es en este punto donde la memoria virtual se hace presente.

Supongamos que para poder comenzar a ejecutar un proceso sólo es necesario cargar en memoria unos pocos fragmentos[3] iniciales de código y de datos. A este conjunto de fragmentos del proceso, se lo denomina conjunto residente.

Definición	**Conjunto residente**: conjunto de páginas o segmentos de un proceso que siempre están en memoria mientras el proceso se está ejecutando.

Usando una tabla de páginas o de segmentos, el procesador puede determinar si la referencia a memoria se encuentra en memoria principal o no. En caso de que no se encuentre, se genera

[3] Fragmento: página o segmento, depende del esquema utilizado.

una excepción indicando un fallo de acceso a la memoria. El sistema operativo coloca al proceso interrumpido en estado bloqueado y toma el control. El sistema operativo se encarga de cargar en memoria principal el fragmento correspondiente a la dirección lógica que generó el fallo.

Para llevar a cabo este proceso, el sistema operativo realiza una petición de E/S para leer el disco. Mientras dicha operación se completa, otro proceso puede pasar a ejecutarse. Cuando el fragmento solicitado es cargado en memoria principal, se genera una interrupción de E/S para permitir al sistema operativo intercambiar los procesos.

El mecanismo descrito trae aparejado las siguientes ventajas:

1. Pueden mantenerse en memoria principal un número mayor de procesos: esto es posible debido a que sólo se cargan las porciones del proceso que son necesarias.
2. Un proceso puede ser mayor que toda la memoria principal.

Debido a que un proceso sólo se ejecuta estando en **memoria principal**, a la misma se la llama **memoria real**. Pero podemos "percibir" a la memoria potencialmente más grande, utilizando una porción del disco. A esto último se lo denomina **memoria virtual**.

Proximidad y memoria virtual

Supongamos un proceso de gran tamaño, es decir, un programa con mucho código y muchos datos. Si analizamos la ejecución del mismo durante un período muy corto de tiempo, la ejecución se puede acotar a una pequeña sección del programa (por ejemplo una función). Si esto se cumple, sería un desperdicio de la memoria, cargar varios fragmentos del proceso cuando sólo se van a utilizar unos pocos. Entonces cada vez que se produce un fallo de memoria, sería mejor cargar sólo unas pocas porciones del proceso, entre las que se debe encontrar el fragmento deseado.

Esto posibilita que haya más procesos en memoria, y ahorra tiempo debido a que no es necesario intercambiar algunas porciones enviándolas a disco.

Una situación normal es que la memoria se encuentre ocupada con fragmentos de varios procesos, para que el procesador y el sistema operativo tengan acceso a la mayor cantidad de procesos. Cuando se debe traer un fragmento nuevo a la memoria, se debe expulsar otro. Si el sistema operativo expulsa un fragmento justo antes de que vaya a ser utilizado, el mismo deberá ser inmediatamente recuperado. A esto se lo denomina **thrashing.**

Definición	**Thrashing**: el sistema consume la mayor parte del tiempo reemplazando porciones de memoria en lugar de ejecutar instrucciones.

Lo que hace el sistema operativo es tratar de predecir, en base a estadísticas de uso, qué porciones de memoria se deben cargar.. Esta predicción es posible gracias al **principio de proximidad**.

El principio de proximidad indica que las referencias a instrucciones y a los datos dentro de un proceso tienden a agruparse. Por lo tanto, durante un período corto de tiempo, sólo se necesitarán unos pocos fragmentos del proceso.

Paginación

A diferencia de lo visto en paginación sencilla (sin memoria virtual), donde toda la tabla de páginas de un proceso debía ser mantenida en memoria principal, en un sistema que utiliza memoria virtual la tabla de páginas de un proceso no necesariamente debe residir completamente en memoria principal, es decir que algunas páginas también residen en memoria virtual.

Por otra parte, la tabla de páginas de cada proceso modifica su estructura. Se agregan dos bits:

- **P (presente)**: indica si la página se encuentra en memoria.
- **M (modificado)**: indica si la página se modificó desde la última vez que fue cargada. Este flag es útil, llegado el momento de reemplazar la página, para saber si la misma debe ser escrita en disco antes de ser reemplazada.

Esquemas de Tabla de páginas

Dos Niveles

Algunos procesos pueden ser excesivamente grandes, por lo que su tabla de páginas completa no puede mantenerse en memoria principal, sino que debe ser dividida entre la memoria principal y la memoria virtual, en otras palabras debemos paginar la tabla de páginas.

Para manejar esta situación, algunos sistemas implementan un esquema de 2 niveles, en el cual se maneja por cada proceso una tabla llamada “directorio”. El directorio posee una entrada por cada página de la tabla de páginas del proceso.

En un segundo nivel aparece la tabla de páginas de cada proceso donde se relaciona el marco de la memoria principal con cada página. Normalmente el directorio de páginas se diseña para que ocupe lo mismo que una página. El directorio de páginas es lo único que permanece constantemente en memoria hasta que el proceso finaliza.

La ventaja de este esquema es que permite reducir el tamaño de la tabla de páginas de un proceso.

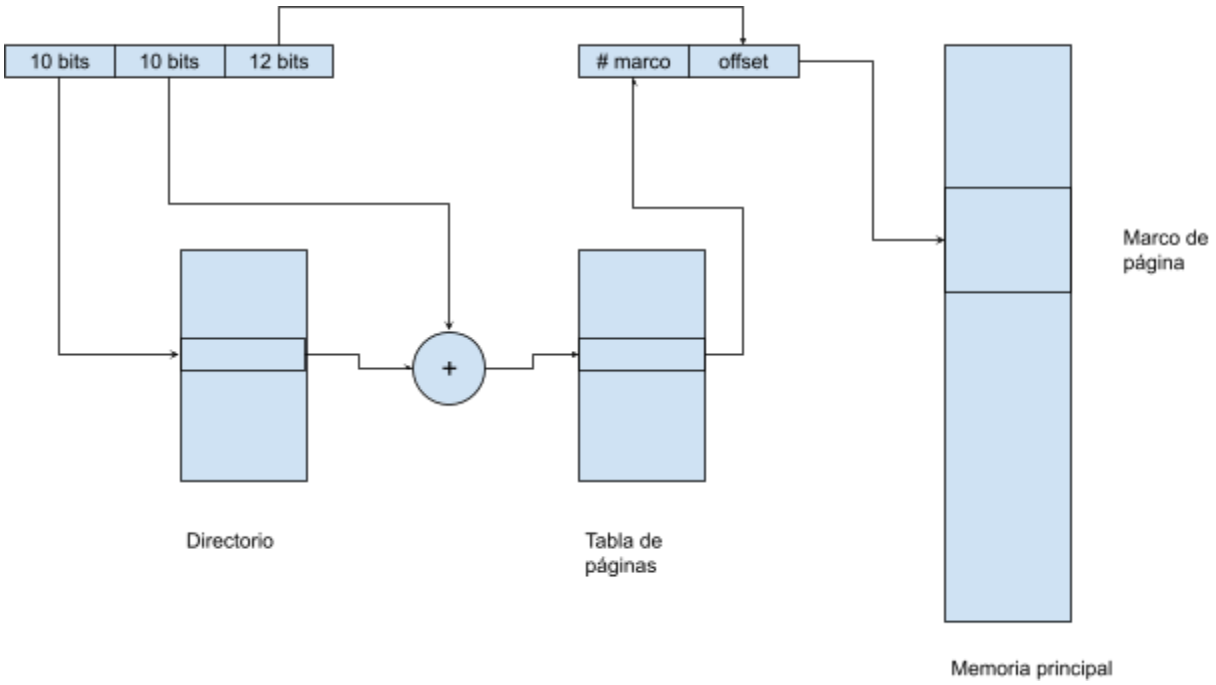
Analicemos esto mediante un ejemplo. Supongamos un sistema con un espacio de memoria direccionable de 32 bits y una página de 4 Kbytes (2^{12} bytes).

Con esos valores, la tabla de páginas de un proceso podría tener 1M de entradas aproximadamente (2^{32} / 2^{12} = 2^{20} = 1.048.576). Manejar una tabla de 1M de entradas es costoso, por lo tanto, si la dividimos en 2 niveles podemos tener una solución de mejor rendimiento y aprovechamiento de la memoria.

Para mejorar esto pensemos lo siguiente:

- Cada dirección de memoria tiene un tamaño de 4 bytes (8 bits * 4 = 32 bits).
- Con un tamaño de página de 4 Kbytes (2^{12} bytes) la tabla de páginas de 1 proceso puede tener hasta 2^{20} páginas, que en memoria ocupan 4 Gbytes (2^{32} bytes).
    - Si cada una de estas 20 páginas, se referencia por una dirección de 4 bytes, la tabla de páginas de un proceso ocuparía 4 Mbytes (2^{20} * 2^2) y para cargar toda la tabla de páginas en memoria (las 2^{20} páginas) necesitamos 2^{10} bytes (2^{22} / 2^{12} = 2^{10} bytes).
- Ahora bien, podemos pensar en una tabla llamada “directorio” de páginas con 2^{10} entradas, una por cada página de la tabla de páginas. Como cada dirección de memoria ocupa 4 bytes, para almacenar 2^{10} entradas, el directorio requiere un espacio de 2^{12} bytes (2^2 * 2^{10}) que equivale al tamaño de 1 página.

Del último punto se deduce que el tamaño del directorio es igual al tamaño de 1 página y llegamos a la conclusión de que en memoria sólo tendremos ocupado 1 página (directorio) más las páginas de la tabla de páginas que se necesiten cargar en memoria para ejecutar el proceso.



Del gráfico anterior vemos que para un espacio de direcciones de 32 bits, una dirección de memoria se conforma de 10 bits para indexar dentro del directorio, 10 bits para indexar dentro de la tabla de páginas del proceso. A partir de ahí se consigue el número de marco al cual se le suma el desplazamiento para obtener la dirección física.

A pesar de tener la ventaja de reducir el tamaño de la tabla de páginas, su principal desventaja es que la cantidad de accesos a memoria puede reducir el rendimiento del sistema.

Lecturas recomendadas

- Paginación en 2 o más niveles. Disponible en: https://www.geeksforgeeks.org/two-level-paging-and-multi-level-paging-in-os/
- Paginación multinivel. Disponible en: https://www.geeksforgeeks.org/multilevel-paging-in-operating-system

Tabla de páginas invertida

En este esquema hay una entrada en la tabla de páginas invertida por cada marco de página real en lugar de por cada página virtual.

La parte correspondiente al número de página de la dirección virtual se referencia por medio de un valor hash.

En este esquema la tabla de páginas siempre requiere una porción fija de la memoria real, independientemente del número de procesos o páginas.

Los campos principales de la tabla de páginas son:

- el Número de Página
- el Id de proceso
- bits de control.

El proceso de traducción de una dirección lógica consiste en:

1. Obtener el número de página de la dirección de memoria y junto con el id de proceso llamar a la función que genera el hash.
2. Con el hash generado acceder a la tabla de páginas invertida para obtener el marco

Buffer de traducción anticipada (TLB)

En principio toda referencia a memoria virtual involucra 2 accesos a memoria, uno para buscar la entrada en la tabla de páginas y otro para buscar los datos.

Para evitar esta situación, la mayoría de los esquemas de memoria virtual implementan una memoria cache especial de alta velocidad para las entradas de la tabla de página. Esta cache se denomina buffer de traducción anticipada (Translation Lookaside Buffer) y contiene aquellas entradas de la tabla de páginas que han sido usadas recientemente.

El proceso de traducción de una dirección lógica es como sigue:

1. El procesador busca la entrada en la tabla de páginas en la TLB.
    a. En caso de encontrarse se recupera el número de marco y se construye la dirección real.
2. Si no se encuentra la entrada en la TLB (fallo de TLB), el procesador accede a la tabla de página. Se recupera el marco y se construye la dirección real.
    a. La entrada de la tabla de páginas accedida será cargada en la TLB, para futuras referencias.
3. Si la referencia no se encuentra en memoria, se produce un fallo de página y el control es cedido al sistema operativo para que cargue la página solicitada.
    a. La entrada de la tabla de páginas accedida será cargada en la TLB, para futuras referencias.

Tamaño de la página

Otro aspecto importante a considerar en cuanto al diseño de un sistema de memoria virtual es el tamaño de cada página.

Este aspecto es importante debido a que a mayor tamaño, menor fragmentación interna. Por otra parte, un tamaño de página grande hace que el *principio de proximidad* pierda fuerza por que habrá referencias a memoria no utilizadas recientemente con lo cual los fallos de página serán frecuentes. Tener en cuenta que si hiciéramos crecer la página a un tamaño tal que pueda contener todo un proceso, no habría fallos de página ni fragmentación interna.

Con un tamaño menor de página habrá muchas más páginas en memoria donde cada una contendrá referencias recientes. Las desventajas son que las tablas de páginas deberán ser más grandes, por lo que habrá mayor cantidad de páginas en memoria virtual. Todo este conjunto de cuestiones nos conduce a la posibilidad de que para una referencia a memoria ocurran 2 fallos de página: el primero por no encontrarse la página de la tabla de páginas y el segundo por no encontrarse la página en memoria principal.

Paginación y segmentación

En un esquema combinado, un proceso se divide en un número de segmentos, donde cada segmento es dividido en un número de páginas de tamaño fijo. Estas páginas son del tamaño de los marcos de la memoria principal.

Desde el punto de vista del programador, una dirección lógica está formada por un número de segmento y un desplazamiento. Desde el punto de vista del sistema, el desplazamiento dentro del segmento es visto como un número de página y un desplazamiento.

Asociado a cada proceso existe una tabla de segmentos y varias tablas de páginas, una por cada segmento.

Políticas del sistema operativo

En todo sistema de administración de memoria se debe evaluar los siguientes 3 puntos:

- ✓ Uso de memoria virtual o no.
- ✓ Paginación, segmentación o ambas
- ✓ Algoritmos utilizados por el sistema operativo para la gestión de la memoria.

Respecto de este último punto, debemos considerar:

1. Políticas de recuperación
2. Políticas de ubicación.
3. Políticas de reemplazo
4. Gestión del conjunto residente
5. Políticas de limpieza
6. Control de carga

Describiremos cada una de estas políticas a continuación.

Políticas de recuperación

Involucra decisiones de cuándo una página debe ser traída a memoria principal. Las 2 alternativas son:

- Bajo demanda: una página se trae a memoria sólo cuando se la referencia. Con esta política cuando un proceso inicia habrá una ráfaga de fallos de página que se irá normalizando a medida que se carguen las páginas referenciadas. Gracias al principio de proximidad los fallos de página tenderán a reducirse
- Paginación adelantada: además de traer a memoria la página buscada, se traen otras. Si las páginas de un proceso se encuentran almacenadas en posiciones contiguas (en memoria secundaria) será más eficiente traer a memoria un número de páginas contiguas de una vez que traerlas de a una.

Políticas de ubicación

Esta política determina en qué parte de la memoria principal van a residir las porciones de la memoria de un proceso. Para sistemas de paginación pura o segmentación más paginación este tema es irrelevante.

Políticas de reemplazo

Cuando todos los marcos de la memoria principal están ocupados, y es necesario traer una nueva página para resolver el fallo de página, esta política determina qué página de las que actualmente se encuentran en memoria va a reemplazarse.

El objetivo es seleccionar la página que tiene menos posibilidades de ser referenciada en un tiempo corto.

Antes de ver los algoritmos utilizados, vamos a comentar que todas las políticas de reemplazo tienen una restricción: algunos marcos de la memoria están bloqueados. Cuando un marco está

bloqueado, la página almacenada en el mismo no puede ser reemplazada. Por ejemplo la mayor parte del núcleo del sistema operativo se carga en marcos bloqueados, buffers de E/S.

Algoritmos básicos

Óptimo

El concepto tras este algoritmo es seleccionar para reemplazar la página para la cuál el instante de la siguiente referencia se encuentra más lejos. Este algoritmo produce la menor cantidad de fallos de página. Este algoritmo es imposible de implementar debido a que el sistema operativo debería conocer los eventos futuros.

LRU

La política *usada menos recientemente* (LRU) seleccionará como página candidata aquella que no se haya referenciado por mucho tiempo. Debido al principio de proximidad, esa página será la que tiene menos probabilidades de volver a ser referenciada en un futuro próximo.

Si bien esta política arroja buenos resultados, similares al algoritmo óptimo, su implementación es muy difícil ya que deberíamos "etiquetar" de alguna forma las páginas con el momento en el que son accedidas para de esa forma realizar la búsqueda al momento de necesitar reemplazarlas.

FIFO

Esta política trata los marcos de página como si fuese un buffer circular y las páginas se reemplazan mediante una estrategia cíclica de tipo *round robin*. Es una de las políticas más simples de implementar. La idea tras este algoritmo es muy simplista, asume que una página traída a memoria hace mucho tiempo, puede haber dejado de utilizarse. Este concepto es erróneo debido a que hay páginas que son cargadas al inicio del programa y referenciadas todo el tiempo. En este caso, esas páginas son expulsadas y traídas de vuelta a la memoria todo el tiempo.

Si bien este algoritmo es muy simple de implementar, su rendimiento es muy pobre.

Reloj

Al igual que FIFO, utiliza un buffer circular de páginas, pero a cada marco de página se le agrega un bit adicional denominado *bit de usado*. Cuando una página se trae a memoria, el bit de usado se pone en 1 y en cualquier momento en que la página se referencie, también se pone en 1 el bit de usado.

Cuando se produce un fallo de página, el sistema operativo recorre el buffer circular de marcos de páginas, buscando aquel cuyo bit de usado esté en 0 (cero). Cada vez que encuentra un

marco con el bit de usado en 1 lo cambia a 0 (cero) y continúa con el marco siguiente. Tener en cuenta que este algoritmo deja situado el puntero en el marco siguiente al reemplazado, de esta forma en el próximo fallo de página se comenzará el algoritmo a partir de ese marco.

Algunos ejemplos:

Flujo de páginas	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					fallo		fallo			fallo		

Flujo de páginas	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
					fallo		fallo		fallo	fallo		

Flujo de páginas	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	5	5	5	5	3	3	3	3
		3	3	3	3	2	2	2	2	2	5	5
				1	1	1	4	4	4	4	4	2
					fallo	fallo	fallo		fallo		fallo	fallo

Flujo de páginas	2	3	2	1	5	2	4	5	3	2	5	2
RELOJ	2^0	2^0	2^0	2^0	5^0	5^0	5^0	5^0	3^0	3^0	3^0	3^0
		3^0	3^0	3^0	3^1	2^0	2^0	2^0	2^1	2^0	2^1	2^0
				1^0	1^1	1^1	4^0	4^0	4^1	4^1	5^0	5^0
					fallo	fallo	fallo		fallo		fallo	

Reloj mejorado

Este algoritmo es igual a su antecesor, sólo que además del bit de usado agrega un *bit de modificado* el cual se pone en 1 si la página dentro del marco en cuestión fue modificada desde que se trajo a memoria.

Con este nuevo bit, podemos tener las siguientes combinaciones sobre los marcos y sus páginas:

	Usado	Modificado
No se accedió recientemente ni se modificó	0	0
No se accedió recientemente pero se modificó	0	1
Se accedió recientemente pero no se modificó	1	0
Se accedió recientemente y se modificó	1	1

El algoritmo de reemplazo se modifica como se indica a continuación:

1. Comenzando por la posición actual del puntero, recorremos el buffer de marcos. No se cambia ningún bit de usado. El primer marco con usado = 0 y modificado = 0 se selecciona para reemplazo.
2. Si el paso 1 falla, se busca el primer marco usado = 0 y modificado = 1. A medida que se recorren los marcos se cambia el bit de usado a 0 (cero)
3. Si el paso 2 falla, el puntero regresó al marco inicial y todos los marcos tienen el bit de usado en 0 (cero). Se repite el paso 1, con lo que se debe encontrar un marco.

La ventaja de este algoritmo es que da preferencia para el reemplazo a las páginas que no fueron modificadas, de forma tal que se evita una escritura en disco.

Buffering de páginas

Este algoritmo guarda ciertas similitudes con el algoritmo FIFO, sólo que para mejorar su rendimiento se definen 2 listas: una para páginas modificadas y otra para páginas no modificadas. Ante un fallo de página, este algoritmo no elimina inmediatamente de la memoria una página sino que la asigna a la lista que corresponda.

En todo momento este algoritmo se reserva un conjunto de marcos libres sobre los cuales implementa las listas de páginas mencionadas.

Cuando se produce un fallo de página, si la página a reemplazar no fue modificada, se mueve a la lista de páginas no modificadas. Por el contrario, si la página fue modificada se mueve a la lista de páginas modificadas. Note que la página si se elimina de la tabla de páginas del proceso, pero se mantiene en memoria. Esto da la ventaja de que si la página vuelve a ser referenciada no hay que ir a buscarla a disco.

Éstas listas de páginas funcionan como una cache de páginas. Cuando se deben escribir páginas a disco, se escriben todas las páginas al mismo tiempo. Esto reduce de forma significativa el número de operaciones de E/S y por lo tanto los tiempos de acceso a disco.

Gestión del conjunto residente

Esta política aborda las siguientes cuestiones:

1. ¿Cuántos marcos de página se van a reservar para cada uno de los procesos activos?
2. El conjunto de páginas que se va a considerar para realizar el reemplazo se limita a aquellas del mismo proceso que causó el fallo de página o se consideran todos los marcos de páginas de la memoria principal.

Para el primero de los puntos se analizan 2 alternativas, asignación fija y asignación variable.

La asignación fija determina un número fijo de marcos de página para cada proceso, que se determina al momento de cargar el proceso en base al tipo de proceso o a guías proporcionadas por el administrador de sistema. Con la política de asignación fija, cada vez que se produce un fallo de página, la página que se necesite reemplazará a una página del proceso.

La asignación variable permite que el número de marcos del proceso cambie durante el tiempo de ejecución del mismo. Si un proceso está generando muchos fallos de página, el sistema operativo podría asignarle marcos de página extra para disminuir la tasa de fallos de página, mientras que un proceso con una tasa de fallos de página baja, se podrá reducir la cantidad de marcos asignados.

El segundo punto trata el concepto de ámbito de reemplazo, que puede ser local (al proceso) o global (todos los procesos).

Una política de reemplazo local sólo podrá seleccionar páginas residentes del proceso que generó el fallo de página, mientras que una política global considerará todas las páginas que se encuentran en memoria principal independientemente del proceso al que pertenezcan.

Resumiendo, combinando los 2 conceptos sólo podemos tener los siguientes esquemas:

- Asignación fija, ámbito local

- Asignación variable, ámbito global
- Asignación variable ámbito local.

Políticas de limpieza

Esta política determina cuando una página que ha sido modificada debe ser escrita nuevamente en disco. Existen 2 alternativas: limpieza bajo demanda y limpieza adelantada.

Con *limpieza bajo demanda* una página modificada sólo es escrita en disco cuando fue seleccionada para ser reemplazada. Si bien esta técnica minimiza las escrituras de páginas obliga al proceso que produjo el fallo de página a esperar que se completen 2 operaciones de E/S, 1 la de escritura de la página a reemplazar y 2 la carga en memoria principal de la página solicitada.

Por otra parte, con *limpieza adelantada* se escriben las páginas modificadas antes de que sus marcos de páginas se necesiten. La ventaja de esta técnica es que permite la escritura en lotes de las páginas modificadas minimizando las operaciones de E/S.. La desventaja es saber y balancear qué páginas conviene escribir en disco, ya que no tiene sentido escribir una página en disco si va a ser modificada nuevamente antes de su reemplazo.

Una buena alternativa a los problemas de ambas políticas es implementar el algoritmo de *buffering de páginas*.

Control de carga

El control de carga determina el grado de multiprogramación del sistema, es decir, determina el número de procesos que residirán en la memoria principal.

Este aspecto es clave para una gestión efectiva de la memoria. Si la cantidad de procesos es muy baja, hay más posibilidades de que en un momento determinado todos los procesos estén bloqueados y el procesador ocioso. Si por el contrario, la cantidad de procesos es muy alta el tamaño del conjunto residente no será adecuado y los fallos de página serán frecuentes.

El grado de multiprogramación puede incrementarse o reducirse.

Estudios realizados demuestran que si se logra igualar el tiempo medio entre los fallos de páginas al tiempo que demora procesar un fallo de página, entonces se logrará una utilización máxima del procesador. Este estudio fue enunciado por Denning y se lo conoce como L = S.

Otra alternativa es adaptar el algoritmo del reloj para que utilice un ámbito global de selección de páginas. La modificación hace que se monitoree la tasa de fallos de página. Si la tasa está por debajo de un nivel determinado entonces el grado de multiprogramación puede incrementarse

con seguridad. Por el contrario, si la tasa de fallos de página supera dicho nivel, entonces el grado de multiprogramación es muy alto.

Para reducir el grado de multiprogramación, lo que se debe realizar es suspender procesos, es decir, enviar procesos a swap. Una vez más tenemos distintas alternativas, suspender:

- los procesos con baja prioridad
- los procesos que provocan muchos fallos de página: esto puede deberse a que el conjunto residente del proceso no sea el adecuado.
- el proceso activado hace más tiempo.
- el proceso con el conjunto residente de menor tamaño: esta estrategia se sustenta en el hecho de que el esfuerzo para volver a cargar el proceso será mínimo.
- el proceso con mayor cantidad de marcos: ésta estrategia busca obtener la mayor cantidad de marcos libres en memoria principal.
- el proceso con la mayor ventana de ejecución restante: ésta estrategia se aproxima al algoritmo de planificación "*primero el proceso con menor tiempo de ejecución"*

Bibliografía utilizada

● William Stallings. Sistemas operativos. Pearson Education. S.A., Madrid, 2005. ISBN-84-205-4462-0.

● AndrewS.Tanenbaum. Modern Operating System. Pearson Education Inc., 2009. ISBN-Q-lB-filBMST-L

● Multilevel Feedback queue. Wikipedia, La enciclopedia libre, 2019 [consulta: 21 de marzo del 2019]. Disponible en https://en.wikipedia.org/wiki/Multilevel_feedback_queue