SMRP: Una aplicación informática para simular el reemplazo de páginas de memoria en los Sistemas Operativos

Carlos Ernesto Velázquez Rodríguez

Carrera: Ingeniería informática.

Instituto Superior Minero Metalúrgico (Cuba).

Resumen: El artículo describe la aplicación SMRP (*Simulador de Módulo de Reemplazo de Páginas*) que simula el módulo de reemplazo de páginas de un sistema de gestión de memoria de un Sistema Operativo y que implementa tres algoritmos (FIFO, LRU y LFU). En esta publicación se realiza una breve incursión sobre el basamento teórico del tema de la administración de memoria, el esquema de memoria virtual, la técnica de paginación y los algoritmos de reemplazo de páginas.

Palabras clave: Sistemas Operativos; administración de memoria; memoria virtual; paginación; algoritmo de reemplazo de páginas; simulación; informática.

SMRP: A computer application to simulate page replacement algorithms for memory pages in computer operating systems

Abstract: The article describes an SMRP application (Memory Page Replacement Simulation Program); which simulates the module of replacing pages from the memory management system in an operating system and implements three algorithms (FIFO, LRU & LFU). Memory management has become a very important topic within the ongoing research on operating systems. This article also provides a brief overview of the theory of memory management, paging techniques and page replacement algorithms.

Key words: Operating systems; memory management; virtual memory; paging; page replacement algorithms; simulation, computers science.

Introducción

En el año 1991, las organizaciones internacionales ACM *(Association of Computing Machinery)* e IEEE *(Institute for Electrical and Electronic Engineers)* publicaron en colaboración un conjunto de condiciones, entre ellas las materias imprescindibles que mínimamente deben observarse en los planes de estudio de las carreras afines a las Ciencias Informáticas. Estas recomendaciones han sido tomadas en cuenta en todas las universidades de Estados Unidos, y de alguna u otra forma, en las europeas e hispanoamericanas (Pressman, 1997).

Uno de los campos que deben cubrirse en estas carreras es el relacionado a los Sistemas Operativos (SO), asignatura que se imparte en el plan de estudios de la carrera de Ingeniería Informática en el Instituto Superior Minero-Metalúrgico de Moa (ISMMM), acatando las recomendaciones de los organismos profesionales internacionales.

En el desarrollo de los sistemas de administración de memoria, el esquema más eficiente y eficaz al que se ha arribado hasta la actualidad para manejar este recurso implica el mecanismo de Memoria Virtual (MV). El concepto presente, tras esta modalidad, es que los programas que se ejecutan en una computadora pueden exceder el tamaño de la memoria física disponible en esa máquina. Haciendo uso de una técnica llamada *paginación*, el SO mantiene en disco aquellas partes del programa que no se utilizan, y en la memoria principal las secciones que sí están en uso (Tanenbaum, 1987). Esta forma de administrar la memoria se complementa con la multiprogramación (Fotheringham, 1961).

En su ejecución, los programas producen un conjunto de direcciones de memoria (eg: en instrucciones donde se les asignan valores a variables). Estas direcciones, conocidas como virtuales o lógicas, deben ser transformadas a referencias de direcciones reales en la memoria física, por un componente del SO llamado Unidad de Administración de Memoria (MMU, por sus siglas en inglés).

Lo que sucede, en pocas palabras, es que cuando se genera una de estas direcciones virtuales, la MMU analiza a qué página de la MV hace referencia. En ese momento solo pueden pasar dos cosas:

1. Que la página tenga asociado un espacio o marco en la memoria física real y entonces se ejecute el fragmento de programa correspondiente.
2. Que la MMU determine que la página no tiene asociado ningún marco, y es cuando ocurre un *fallo de página*, una indicación de qué asignársele a esa página uno de los marcos de la memoria física.

Si quedaran marcos libres, no existe mucho problema en otorgarle a la página en cuestión uno de esos que aún están disponibles, pero ¿qué sucede cuando todos los marcos están ocupados? En ese caso, la MMU debe decidir qué página de las que están en memoria física despojará del marco que usa para concedérselo a la página que provocó el fallo.

El *cómo* decidir a qué página se le retirará su marco es de lo que se encargan los algoritmos de reemplazo de páginas, sobre los que se han escrito una amplísima colección de artículos. Puede encontrarse una gran parte de ellos en Smith (1978).

Los algoritmos de reemplazo de páginas buscan minimizar la ocurrencia de los fallos de página para elevar el rendimiento del desempeño del sistema. Todo en las ciencias de la informática trata de agilizar, de hacer más rápidos los cómputos que se llevan a cabo. Tanenbaum (1987) explica en detalle lo que ocurre al producirse un fallo de página:

1. *El hardware hace un señalamiento al núcleo y resguarda el contador del programa en pila. En muchas máquinas, se resguarda cierta información acerca del estado de la instrucción activa en algunos registros especiales de la CPU.*
2. *Inicia una rutina en código ensamblador con el fin de resguardar los registros generales y la demás información volátil y evitar que el SO los destruya. Esta rutina llama al SO como un procedimiento.*
3. *El SO descubre la ocurrencia de un fallo de página e intenta determinar la página virtual requerida. Con frecuencia, uno de los registros de hardware contiene esa información. Si no, el SO debe recuperar el contador del programa, buscar la instrucción y analizarla en software para hacerse una idea de lo que realizaba al ocurrir el fallo.*
4. *Una vez determinada la dirección virtual que provocó el fallo, el SO verifica si la dirección es válida y si la protección es consistente con el acceso. Si no, se envía una señal al proceso o se elimina. Si la dirección es válida y no ocurrió un fallo de*

protección, el sistema intenta obtener un marco de la lista de marcos libres. Si no existen marcos libres, se ejecuta el algoritmo de reemplazo para seleccionar una víctima.

5. *Si la página seleccionada está sucia, el planificador transfiere la página al disco y ocurre un cambio de contexto; se suspende el proceso que provocó el fallo y se permite la ejecución de otro, hasta que termina la transferencia al disco. En cualquier caso, el marco queda señalado como ocupado, para evitar que se le use con otros fines.*
6. *Tan pronto el marco está limpio (ya sea en forma inmediata o después de ser escrito en disco) el SO examina la dirección en disco donde se encuentra la página necesaria y planifica una operación en disco para recuperarla. Mientras se carga la página, el proceso que produjo el fallo continúa suspendido y se ejecuta otro proceso del usuario, si está disponible.*
7. *Cuando el interruptor del disco indica la llegada de la página, las tablas de página se actualizan para reflejar su posición y el marco queda en estado normal.*
8. *La instrucción que produjo el fallo regresa al estado donde se inició y el contador del programa se modifica para que apunte a esa instrucción.*
9. *EL proceso que provocó el fallo se planifica y el SO regresa a la rutina del lenguaje ensamblador que lo llamó.*
10. *Esta rutina restaura los registros y la demás información volátil; por último, regresa al espacio del usuario para continuar la ejecución, como si no hubiera ocurrido un fallo de página.*

Lo que simula SMRP es una abstracción de una pequeña parte de todo el proceso que realiza la MMU para responder ante un fallo de página, a través de tres de estos algoritmos (FIFO, LRU y LFU). El software llamado *"Simulador de Módulo de Reemplazo de Páginas"*, o SMRP, muestra el estado de la memoria física luego de ser atendida cada solicitud de una lista de peticiones de páginas, además de indicar la cantidad de fallos que generan para esa lista de accesos el algoritmo seleccionado con el número de marcos predefinido.

Principio básico de funcionamiento de la memoria virtual

Como se mencionó con anterioridad, la memoria virtual (MV) es un esquema de administración de memoria diseñado por Fotheringham (1961), que permite que programas que excedan el tamaño de la memoria física de una computadora puedan

ejecutarse como si entraran perfectamente en ella. El Sistema Operativo (SO) se vale de una técnica llamada *paginación*, para dividir en pequeños módulos la imagen del programa que permanece en el disco, mientras ejecuta parte a parte cada una de estas secciones de la aplicación, manteniendo en memoria el fragmento que está corriendo, y en disco las partes que no. Estos ardides permiten, por ejemplo, que en nuestras computadoras de 4 GB de memoria RAM podamos ejecutar aplicaciones tan potentes como juegos de 7, 10 o 12 GB. Sin dudas, es un acertado mecanismo que ha permitido a los programadores de las empresas de sofware incrementar el potencial uso de la memoria.

¿Cómo funciona la paginación?

La mayoría de los SO que usan MV se valen de la paginación para explotar las bondades del esquema de Fotheringham. Básicamente, los programas, en cualquier ordenador, durante la ejecución generan un conjunto de direcciones lógicas o virtuales, *e.g.: int valor = 4*. En estas computadoras existe un conjunto de chips que conforman la Unidad de Administración de Memoria, que relacionan estas direcciones virtuales a direcciones físicas reales.

El total de direcciones lógicas que se pueden generar conforman el espacio de direcciones lógicas, y se puede dividir en pequeñas unidades llamadas *páginas*. La memoria física se atomiza en partes del mismo tamaño que las páginas, pero denominadas *marcos*. Las *páginas* de la memoria lógica se asocian a marcos de la memoria física, que es donde finalmente se ejecutan las aplicaciones.

Cuando una aplicación genera una dirección virtual, la MMU la toma, determina qué página está solicitando la dirección generada, obtiene el marco al que está asignada la página solicitada y carga los datos en esa región de la memoria física.

Pero sucede que la memoria física es mucho menor que la disponibilidad de la memoria virtual, y esto implica que en cada instante de tiempo, sólo un pequeño grupo de páginas tendrán asignados todos los marcos, uno para cada una.

Por esta razón puede suceder, y de hecho sucede comúnmente, que la página a la que la dirección generada hace referencia no tenga asignado ningún marco. En ese momento, la MMU hace un señalamiento al SO a través de la CPU, indicando un fallo de página.

El SO entonces pone en marcha un procedimiento que se encarga de tomar un marco de la memoria física para asignárselo a la página que provocó el fallo. En el caso de que queden marcos libres, sencillamente se le otorgará uno de esos. Sin embargo, cuando todos los marcos están ocupados habrá que despejar alguna página de las que los tienen en uso.

Es necesaria una alta tasa de fallos de página para elevar el rendimiento del desempeño del sistema. Por ello es aconsejable escoger una página de poco uso, de las que están en memoria, para quitarle su marco y otorgárselo al acceso que hizo que la CPU señalara al SO el error. La cuestión es que existen varias maneras de resolver este procedimiento.

La forma en que el módulo de reemplazo de páginas (la parte del SO que decide qué página despojar de su marco cuando ha ocurrido un fallo) escoge a su víctima es un algoritmo de reemplazo de páginas, de los que existen muchos.

Políticas o algoritmos de reemplazo de páginas

Al ocurrir el error de página, si todos los marcos están ocupados, el módulo de reemplazo en el SO debe elegir una página para eliminarla de la memoria física y crear el lugar para la petición que creó el fallo.

Aunque, simplemente, es muy fácil escoger aleatoriamente un marco de página, el rendimiento del Sistema Operativo mejora considerablemente si se siguen determinadas recomendaciones para atrasar o minimizar la ocurrencia de los fallos. La aplicación descrita en este artículo provee tres de los que se estudian en las carreras afines. Sin embargo, antes de dedicarnos a enunciar el comportamiento teórico de estas tres políticas, explicaremos algunos detalles concernientes a la cota máxima del rendimiento de estas. Se trata del algoritmo óptimo.

Política de reemplazo de Páginas Óptima

Según palabras de Tanenbaum (1987) "el mejor algoritmo es fácil de describir, pero imposible de implantar". El principio es el siguiente: si conocemos qué páginas serán referenciadas en el futuro, al momento de ocurrir un fallo evitaremos tomar alguna de estas páginas para desalojarlas de su marco, dado que de hacerlo, esta acción conllevaría a nuevos fallos más tarde.

El problema es que los implementadores no pueden algoritmizar un proceso tan "trivial" como predecir el futuro. Es imposible hacerlo, al menos por ahora. De cualquier forma no es insalvable la adivinación del destino, y podemos hacer un poco de trampa. Es posible simular la predicción si ejecutamos un algoritmo que recorra toda la aplicación que se practicará, etiquete con el número de la instrucción del programa la página a la que se referencia en esa línea, y finalmente, use esos valores en la ejecución del algoritmo óptimo para ‹predecir el futuro›, a través de esa información obtenida de antemano.

Este algoritmo puede ser usado para establecer una comparación de las demás políticas con respecto al mejor algoritmo posible, pero sólo para el mismo programa que genere la misma secuencia de solicitudes de páginas.

La política óptima no es del todo impracticable y además tiene un fin que, contraproducentemente, es muy práctico: evaluar la eficiencia de los demás algoritmos que sí son aplicables a sistemas reales.

Establecidos los pilares que soportan la naturaleza de la eficiencia de estos algoritmos, nos disponemos a exponer una breve descripción de las políticas de reemplazo que hemos implementado en nuestra aplicación.

Breve descripción del algoritmo FIFO

FIFO significa ‹*First In, First Out*›, es decir, el primero en entrar es el primero en salir. Este algoritmo es el mismo que se sigue en cualquier tipo de cola sin prioridades. Las peticiones se organizan en el orden de llegada y se atienden en ese mismo sentido.

Este algoritmo supone que la petición más antigua es la que menos se necesitará en el futuro, y por esta razón es la que eliminará cuando deba decidir, de las páginas que están cargadas, a cuál quitarle el marco en la que se encuentra, para cedérselo a otra.

En la realidad, FIFO, en su forma pura, no es una buena estrategia para evitar una alta tasa de fallos de página, y es por esta razón que no se aplica estrictamente en los sistemas que lo implementan de alguna manera.

Breve descripción del algoritmo LRU

Bajo el supuesto de que la página que más recientemente se ha referenciado es la que más probablemente se referenciará en el futuro cercano, LRU, que formalmente significa ‹*less recently used*› (menor uso reciente), lleva un contador de tiempo para cada página en memoria, que se incrementa en cada instante de tiempo que ha pasado desde que la página se referenció por última vez.

Al momento de producirse un fallo de página, el algoritmo despoja de su marco a la página que tiene mayor valor en el contador de tiempo de referencia.

Tanenbaum (1987) explica que LRU es realizable en teoría (refiriéndose a la implementación en sistemas reales), pero no es barato, y da una serie de explicaciones sobre variantes para implementarlo tanto en *hardware* como en *software*.

Breve descripción del algoritmo LFU

Suponiendo que la página que más ha sido usada en el pasado será la que más se usará en el futuro, LFU ‹*less frequently used*› (menor uso frecuente) gestiona los fallos de página otorgándole al acceso que lo haya producido, el marco de la página que menos frecuencia lleva en uso. Para ello se auxilia de un contador de referencias de cada página en memoria.

Se ha implementado LFU para que decida aleatoriamente cuál de los candidatos que puedan quedar escogerá para sustituir por el acceso que produjo el fallo, y no para que simplemente decida por FIFO. Esto se debe a que, en muchos casos, dependiendo de la lista de peticiones, LFU puede comportarse exactamente como LRU si ambos deciden, por la política FIFO, cómo deben escoger la víctima de sus candidatos.

SMRP. Un acercamiento

Simulador del Módulo de Reemplazo de Páginas (SMRP) es una aplicación de escritorio construida con el IDE NetBeans 7.0, usando el lenguaje de programación Java. Como ha quedado claro en secciones anteriores, permite conocer el estado de la memoria física de un ordenador hipotético que, como parte de su Sistema Operativo, implementa un módulo de reemplazo de páginas de memoria.

De la realidad a la abstracción. Plausibilidad de SMRP

Cuando abordamos el tema de la MV, establecimos que las direcciones virtuales generan las páginas solicitadas mediante la MMU, a través del tiempo, lo que produce una lista de peticiones de las páginas que se cargarán en memoria.

En SMRP es el usuario el que establece directamente la secuencia de solicitudes, así como la cantidad de marcos en la memoria física y el algoritmo de reemplazo de página (de los tres que se ofrecen) que decide usar para su lista de peticiones. Estas facilidades se deben a que SMRP está concebido para servir de ayuda a los estudiantes de Informática que se enfrentan a la asignatura Sistemas Operativos. SMRP es una herramienta automatizada que les brinda la posibilidad de confrontar sus propias ejecuciones de las políticas de reemplazo con la ejecución de un sistema que las implementa.

Esta aplicación también le permite al profesor comprobar las soluciones de ejercicios sobre este tema que realicen sus estudiantes, extendiendo el uso de las TICs a la enseñanza de la asignatura y elevando el nivel científico-técnico del proceso de impartir la materia, aumentando la calidad de la misma y apoyando las recomendaciones de los organismos internacionales rectores en el campo de las ciencias de la informática. La universidad, la facultad y la asignatura hacen honores a las directivas de más alto nivel en el mundo.

¿Qué podemos incluir o mejorar en una nueva versión?

En próximas versiones de SMRP se podría incluir funcionalidades que permiten generar secuencias aleatorias de tamaño variables de peticiones de páginas; ofrecer la posibilidad de guardar en archivos las ejecuciones realizadas; agregar nuevas políticas de reemplazo; diversificar la entrada de los datos, *e.g.:* permitir la entrada de una secuencia de direcciones lógicas para que el sistema las traduzca a direcciones físicas; permitirle al usuario especificar la longitud en bits de las direcciones lógicas, la cantidad de páginas, el tamaño de la memoria física y su tamaño de marco (y de página) para que la aplicación realice también el trabajo de la MMU y el usuario pueda comprender todo el proceso; incluir la ejecución de la política óptima para evaluar el desempeño de los demás algoritmos respecto a una misma secuencia de solicitudes. Son muchas las puertas que abre esta primera versión de SMRP.

Conclusiones

SMRP es una aplicación que implementa el reemplazo de páginas de memoria virtual por tres algoritmos diferentes (FIFO, LRU y LFU). Se apoya en el basamento teórico de la administración de memoria mediante el esquema de la memoria virtual, que se sustenta en la técnica conocida como paginación.

Es una herramienta que sirve de ayuda a estudiantes y profesores de la asignatura Sistemas Operativos (una materia recomendada por la ACM y la IEEE para su enseñanza en las carreras afines a las ciencias informáticas) para confrontar resultados y evaluar el aprendizaje de las políticas de reemplazo de páginas en el tema de la administración de memoria. Las funcionalidades que se pueden agregar en próximas versiones elevarían su valor y flexibilidad, agregando mayor plausibilidad a este sistema.

Referencias bibliográficas

FOTHERINGHAM, J. 1961: Dynamic Storage Allocation in the Atlas Computer Including an Automatic Use of a Backing Store. *Commun. of the ACM* 4(10): 435-436.

PRESSMAN, R. S. 2005: *Ingeniería de Software: Un enfoque práctico* [en línea]. Consultado: 10 dic 2011. Disponible en: http://www.taringa.net/posts/ciencia-educacion/12689892/Ingenieria-del-Software---Roger-Pressman-6ta-Edicion.html

SMITH, A. J. 1978: Bibliography on Paging and Related Topics. *Operating Systems Review* 12: 39-56.

TANENBAUM, A. & WOODHUL A. 1999: *Sistemas Operativos, Diseño e Implementación.* Segunda edición. Editorial Pearson, México.