**Proyecto final Sistemas Operativos, Enero-Mayo 2015**

**Objetivo:** Ejercicio de programación para mostrar su comprensión de algunos de los algoritmos de administración de memoria vistos en clase

**Procedimiento:** Desarrollar un programa en cualquier lenguaje con las siguientes especificaciones:

1. El programa deberá simular un manejador de memoria virtual utilizando paginación *parecido* al visto en clase, y medir su rendimiento. A cada equipo de alumnos le será asignada una estrategia de reemplazo de páginas como las vistas en clase, o bien alguna nueva.
2. Para la simulación, su programa contará con las siguientes estructuras de datos:
   1. Un área de memoria contigua de 2048 bytes de longitud que simulará la parte de la memoria real de una computadora reservada a marcos de página. *Por ejemplo*, puede ser un vector llamado **M** de 2048 elementos, cada uno de 1 byte. Los marcos de página tienen una longitud de 8 bytes; es decir, la memoria consiste de 256 marcos de página. La primera posición del arreglo, y por tanto del primer marco de página, representa la dirección 0 de la memoria real. La última dirección sería la 2047. Inicialmente la memoria está vacía.
   2. Una segunda área de memoria contigua, que simulará el área de disco reservada para el swapping de páginas. *Podría ser por ejemplo* un vector llamado **S** de 4096 elementos, cada uno de 1 byte.
   3. Otras estructuras de datos que encuentren Uds. convenientes o indispensables para lograr la simulación y que se encuentran en áreas de memoria distintas del área **M.**
3. INPUT: Simulará las peticiones que hace el sistema operativo a su propio algoritmo manejador de memoria virtual, tales como asignar o liberar espacios de memoria o accesar direcciones virtuales para obtener la dirección en memoria real.

Los datos de entrada al programa consistirán de un archivo en disco con varias líneas que representan estas solicitudes Cada línea del archivo puede tener uno de los siguientes formatos. Para cada formato, se indica cuál sería *parte* del PROCESS y el OUTPUT requeridos. Infieran Uds. qué estructuras de datos van a necesitar para cumplir con estas especificaciones.

* 1. P n p (cargar un proceso)  
     Se trata de una solicitud de “n” bytes para cargar un proceso a la memoria.

“p” es un número entero que indica el identificador de proceso. Ejemplo:

**P 534 5 (Se solicita asignar 534 bytes al proceso 5)**  
  
NOTA En esta versión de memoria virtual con paginación, un proceso se carga completito a memoria real en una sola solicitud al programa, incluyendo páginas que contienen el código del proceso y otras para su área de datos. La longitud máxima de un proceso es de 2048 bytes. *Observen que esto es diferente a la forma como se revisó en clase, donde aprendimos que los procesos se pueden cargar a memoria por partes en base a un esquema por demanda.*  
PROCESS: El programa deberá asignar los marcos de página necesarios, *no necesariamente contiguos, pues podría no encontrarse un área contigua de la longitud requerida.* En el ejemplo anterior, 534 bytes requieren 67 marcos [(534/8) = 66.75)]. Si no hay marcos de página vacíos suficientes en **M***,* se tendrá que swappear-out las páginas contenidas en algunos marcos ya ocupados, de acuerdo con el algoritmo de reemplazo que se les haya sido asignado. *Las páginas a reemplazar podrían pertenecer a cualquier proceso; es decir, se utiliza “reemplazo global”*. Si hay swapping, su programa deberá registrar dónde queda cada página en el área **S** de swapping, pues posteriormente puede ser necesario volver a cargar esas páginas. Además puede ser necesario registrar el “*timestamp”* u otra información como podría ser el *“bit de referencia”* y *“bit de modificación”* al momento que se asigna o se utiliza en memoria cada página del proceso, en concordancia con lo que requiera la estrategia de reemplazo que se esté utilizando.   
  
*Hint.* Para facilitar su debugging, podrían registrar en las áreas **M** y **S** el número del proceso que está cargado en cada elemento de esas memorias. Esto no es indispensable.  
*Hint 2* Todos los lenguajes tienen librerías que incluyen alguna función para obtener el timestamp; busca *[timestamp <lenguaje que usaste>]* pero ten cuidado que la respuesta que obtienes de Google sea compatible con la versión del lenguaje que estás usando, pues a veces las librerías de los lenguajes cambian con la versión.  
  
OUTPUT: Imprimir el comando de INPUT; la lista de marcos de página asignados. Si fue necesario hacer swapping, una lista de los marcos de memoria real que fue necesario swappear para hacer hueco; para cada uno, indicar, acerca de la página que contenía el marco, a qué proceso y número de página del proceso pertenecía, y dónde quedó esa página en el área de swapping. Se muestran ejemplos del output más adelante.

* 1. A d p m

Es una solicitud para accesar la dirección virtual “d” del proceso “p”. Si “m” es 0, la dirección correspondiente sólo se lee; si es 1, también se modifica.

“d” puede tener un valor desde cero hasta la dirección virtual máxima del proceso.   
Ejemplo:

**A 9 5 0 (accesar para lectura la dirección virtual 9 del proceso 5)**  
PROCESS: Localizar dónde se encuentra la página que contiene esa dirección. En el ejemplo, se trata de la página 1 del proceso, pues la longitud de página es 8 y las páginas de un proceso se numeran desde cero. Esa página 1 puede estar en algún lugar de la memoria real, y no necesariamente junto a la página 0; o bien puede estar en el área de swapping pues ya la habíamos “corrido” de su marco en memoria real en alguna operación anterior. Si ese es el caso, ha ocurrido un “*page fault”*, y es necesario cargarla de nuevo (swapping-in) en alguna parte de la memoria real, para lo cual podría ser necesario swappear-out algún otro marco de página si es que todos están ocupados (no hay marcos libres). Nuevamente, podría ser necesario registrar en alguna parte el *“timestamp”* en que ocurre el acceso o alguna otra información pertinente.  
  
OUTPUT: El comando de INPUT; la dirección en memoria real correspondiente a la dirección virtual “d” del proceso “p”. Si la página necesaria se encontraba en el área de swapping, indicar donde estaba en el arreglo S, y donde quedó en memoria real; si fué necesario para cargarla swappear-out alguna otra página de algún proceso de acuerdo al algoritmo de reemplazo, indicar ésto de manera similar a como se describe en el inciso a)

* 1. L p

Liberar las páginas del proceso “p”.

PROCESS: Se liberan todas las páginas del proceso “m”, tanto las que estaban en memoria real como aquellas que se encontraban en el área de swapping, quedando varios marcos de página o pedazos del área de swapping vacíos y disponibles para otras operaciones.

OUTPUT: El comando de INPUT; lista de marcos de página que se liberaron.

* 1. F   
     Fin. Es siempre la última línea de un conjunto de solicitudes; pero pueden seguir otras líneas más para otro conjunto de solicitudes.

PROCESS: lo necesario para el OUTPUT.

OUTPUT: El comando de INPUT; reporte de estadísticas, que incluye:

- *turnaround time* de cada proceso que se consideró. Se refiere *al tiempo de ejecución de tu programa,* desde que se comienza a cargar un proceso (P) hasta que se terminan de liberar todas sus páginas (L). Puedes obtenerlo por medio de una diferencia de timestamps.

- *turnaround promedio*.

- *número de page faults por proceso*. Recuerda que un page fault ocurre únicamente cuando un marco de página necesario no se encuentra en memoria real. 🡪 *En esta versión del manejador de memoria virtual, el comando P (cargar un proceso completo a memoria real) no produce page faults.* 🡨

- *número total de operaciones de swapp-out y de swapp-in* que fueron necesarias para todos los procesos.

* 1. E  
     Exit. Última línea del archivo.

PROCESS: se termina el programa.

OUTPUT: El comando de INPUT y mensaje de despedida...

1. Algoritmos de reemplazo: A cada equipo de 4 alumnos le será asignado un algoritmo de reemplazo de los vistos en clase, o bien alguno nuevo. Algunos de estos algoritmos hacen uso de timestamps o de los bits o contadores de referencia y otros no.

Asignación por equipo:

|  |  |
| --- | --- |
| Número de equipo (4 integrantes) | Algoritmo de reemplazo |
| 4.1, 4.4, 4.7, 4.10, 4.13, ….. | FCFS |
| 4.2, 4.5, 4.8, 4.11, 4.14, ….. | LRU |
| 4.3, 4.6, 4.9, 4.12, 4.15, ….. | Bit de modificación y bit de referencia |

Cualquier equipo puede decidir utilizar un algoritmo de reemplazo diferente a los anteriores, previa declaración por parte de ustedes y aprobación de sus instructores del curso.

1. El archivo de entrada que utilizarán todos los equipos será dado a conocer dos días antes de la fecha de entrega. Ya para entonces deberá estar ampliamente probado su programa, de tal forma que no tenga ningún problema al ejecutar este archivo de entrada “sorpresa”.
2. *Lenguaje.* El que gusten. Si conocen Python o Ruby o algún otro lenguaje igualmente poderoso, se les puede simplificar la vida. Si no los conocen queda un tiempo muy corto para aprenderlos y además hacer, probar y documentar el programa, así que mejor utilicen algún lenguaje que ya conozcan.
3. Entregables: 14 de mayo a las 11:59 AM (mediodía): Un reporte con:
   1. Hardware, sistema operativo, lenguaje y versión del lenguaje que utilizaron. Tiempo de ciclo de la computadora usada (en megahertz)
   2. El output
   3. El listado del programa, que deberá utilizar programación modular. Ampliamente documentado mediante comentarios en el mismo código, como sigue. Adapten lo siguiente a la nomenclatura usada por el lenguaje que decidan utilizar.
      1. Los autores del programa
      2. Breve descripción sobre para qué sirve el programa.
      3. El significado de cada variable global o local. Nombres de variable muy claros; sin embargo, el comentario sobre el significado preciso de la variable puede ser necesario.
      4. La función de cada función, rutina, gem, método o como se llame, así como su estado de entrada y estado de salida. Nombres igualmente muy claros. El estado de entrada incluye nombres y valores de variables globales y argumentos al entrar a la función. El estado de salida, lo mismo al terminar la función.
      5. Comentarios de bloques de código que indican qué hace el bloque, si es necesario. Lo mismo para ciertas líneas individuales, si es necesario.
   4. La prueba de fuego acerca de la estructura y claridad de su programa: *la prueba de la bicicleta*. No es necesario pensar mucho para darse cuenta de cómo funciona una bici. Basta mirarla. Lo mismo debe ocurrir al echarle una mirada a su programa --cualquier programa que hagan en su vida. Un programa es una obra literaria, no solo un conjunto de instrucciones. Incluso hubo hace unos años un movimiento llamado [literate programming].  
        
      En una escala de 1 a 100, un programa que funciona vale solo 30. Uno que funciona ante cualquier conjunto posible de datos y genera mensajes entendibles si algo anda mal, 50. Uno que fácilmente se puede comprender cómo funciona s*in recurrir a los programadores originales*, vale 80. Uno que además está tan bien documentado y modularizado que es posible hacerle rápidamente los cambios más probables a futuro, vale 100. Los mejores programas “open source” son de este último tipo, pues precisamente la idea es que cualquier persona pueda entenderlos rápidamente y contribuir a mejorarlos.
   5. Su programa debe ser robusto y defenderse ante cualquier error, por ejemplo de input incorrecto, procesos más grandes que lo permitido, direcciones fuera del tamaño del proceso, área de swapping agotada ETC. No son todos… Les sugerimos probar su programa con input aleatorio…
4. Ejemplo de input y output con el algoritmo de reemplazo: “Reemplaza una página contenida en un marco aleatorio”. No muy efectivo pero muy simple. Los //comentarios no son parte de su output. Todo lo demás sí  
     
   **P 2048 1**  
   Se asignaron los marcos de página 0-255 al proceso 1

//¡se llenó la memoria!

// como es el primer proceso, se cargó en memoria contigua

**A 1 1 0**Dirección virtual: 1. Dirección real: 1

// el proceso 1 comienza a ejecutar...

**A 17 1 1**

// modifica la dirección virtual 17 del proceso 1  
Página 2 del proceso 1 modificada. Dirección virtual: 17. Dirección real: 17

**P 16 2**Se asignaron los marcos de página 5 y 78 al proceso 2   
// (números aleatorios de acuerdo a algoritmo de reemplazo,   
// pues no había ningún marco libre)  
página 5 del proceso 1 swappeada a la posición 0 del área de swapping   
página 78 del proceso 1 swappeada a la posición 8 del área de swapping  
// como el proceso 1 estaba en memoria contigua

// empezando desde 0, los marcos de página coinciden  
// con los números de página del proceso

**A 7 2 0**  
Dirección virtual: 7. Dirección real: 47   
// es la página 0 del proceso 2, que se encuentra en el marco 5

// Este empieza en la dirección 8\*5 = 40.

**A 41 1 0**Dirección real: 641  
Página 80 del proceso 1 swappeada a la posición 16 del área de swapping.  
Se localizó la página 5 del proceso 1 que estaba en la posición 0 de swapping  
y se cargó al marco 80.   
// Posición 0 del swapping queda libre. Como no había espacio en M,   
// para cargar esa página 5 fué necesario antes sacar otra, y resultó  
// aleatoriamente que estaba en el marco 80. Ese marco 80 ya   
// estaba ocupado por otra página del mismo proceso 1 que fue necesario   
// swappear a la posición 16 de **S**   
  
**L 2**Se liberan los marcos de página 5 y 78 que ocupaba el proceso 2

**P 16 3**Se asignan los marcos 5 y 78 al proceso 3

**L 1**Se liberan los marcos 0-4, 6-77 y 79-255 de memoria real que ocupaba el proceso 1

Se liberan las posiciones 8 y 16 del área de swapping que ocupaba el proceso 1

**F**\*Error\* - se quedó cargado el proceso 3   
Fin. Reporte de salida:  
(etc…)

(nueva serie de solicitudes. Se re-inicializa todo antes de proseguir)  
E  
babay